②

NAVAL POSTGRADUATE SCHOOL
Monterey, California

DTIC
S ELECTE
MAR 0 8 1990
cg B
D

THESIS

THE DEVELOPMENT
OF A
RAPID PROTOTYPING ENVIRONMENT

by

Laura J. White

December 1989

Thesis Advisor: Luqi

90 03 07 040

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED		1b RESTRICTIVE MARKINGS			
2a. SECURITY CLASSIFICATION AUTHORITY		3 DISTRIBUTION/AVAILABILITY OF REPORT			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE					
4. PERFORMING ORGANIZATION REPORT NUMBER(S)		5 MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School	6b OFFICE SYMBOL (If applicable) 37	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School			
6c. ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000		7b ADDRESS (City, State, and ZIP Code) Monterey, CA 93943-5000			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION	8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER			
8c. ADDRESS (City, State, and ZIP Code)		10 SOURCE OF FUNDING NUMBERS			
		PROGRAM ELEMENT NO	PROJECT NO	TASK NO	WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

THE DEVELOPMENT OF A RAPID PROTOTYPING ENVIRONMENT (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
White, Laura J.

13a. TYPE OF REPORT Master's Thesis	13b. TIME COVERED FROM _____ TO _____	14 DATE OF REPORT (Year, Month, Day) 1989 December 21	15 PAGE COUNT 361

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government

17.	COSATI CODES		18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Rapid Prototyping, PSDL, CAPS, Graphic Editor, Syntax
			Directed Editor, Translator, Static Scheduler, Software
			Database, Real-Time Software, Embedded Systems

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Currently the development and maintenance of DOD embedded software systems with hard real-time constraints is a very complex, time-consuming and costly task. This situation can be improved by the use of adequate development methods and powerful support tools. This thesis explores the development and integration of rapid prototyping tools for the Computer Aided Prototyping System (CAPS). CAPS supports the design and evolution of large, reliable embedded software systems while significantly reducing their associated development and maintenance costs.

CAPS utilizes the Prototype System Description Language (PSDL) and an integrated set of construction and analysis tools. The integration of these tools utilizes previous work on their design, with partial implementations and feasibility studies for some of the tools. We have defined and implemented a user interface while testing previous tools, refining the designs of the tools and either refining the implementations or generating the initial implementations. The user interface provides systematic access to the tools of the environment

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof Luqi	22b TELEPHONE (Include Area Code) (408) 646-2735	22c OFFICE SYMBOL 52Lq

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—806-24.

19. ABSTRACT Continued:

to support the underlying rapid prototyping methodology. Integration issues include system configuration, integration testing, design modifications, implementations, and evolution of previously developed tools within this rapid prototyping environment.

Accession For	
NTIS GRA&I	☑
DTIC TAB	☐
Unannounced	☐
Justification	

By_____
Distribution/
Availability Codes

Dist	Avail and/or Special

The Development
of a
Rapid Prototyping Environment

by

Laura J. White
Lieutenant, United States Navy
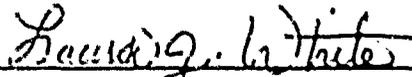B.S., University of New Mexico, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE
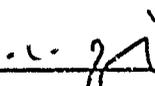
from the

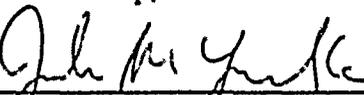NAVAL POSTGRADUATE SCHOOL
December 1989

Author: _____
Laura J. White

Approved by: _____
Luqi, Thesis Advisor

_____
John M. Yurchak, Second Reader

_____
Robert B. McGhee, Chairman
Department of Computer Science

iii

ABSTRACT

Currently the development and maintenance of DOD embedded software systems with hard real-time constraints is a very complex, time-consuming and costly task. This situation can be improved by the use of adequate development methods and powerful support tools. This thesis explores the development and integration of rapid prototyping tools for the Computer Aided Prototyping System (CAPS). CAPS supports the design and evolution of large, reliable embedded software systems while significantly reducing their associated development and maintenance costs.

CAPS utilizes the Prototype System Description Language (PSDL) and an integrated set of construction and analysis tools. The integration of these tools utilizes previous work on their design, with partial implementations and feasibility studies for some of the tools. We have defined and implemented a user interface while testing previous tools, refining the designs of the tools and either refining the implementations or generating the initial implementations. The user interface provides systematic access to the tools of the environment to support the underlying rapid prototyping methodology. Integration issues include system configuration, integration testing, design modifications, implementations, and evolution of previously developed tools within this rapid prototyping environment.

THESIS DISCLAIMER

Ada is a registered trademark of the United States Government, Ada Joint Program Office.

TABLE OF CONTENTS

vi

viii

ACKNOWLEDGEMENTS

Dedicated to Bea? .egard, Buddy and the memory of Wolfgang, special friends who brought much love ' happiness into my life during the period of this thesis.

I would like to express my gratitude to my family and friends for their continual encouragement, guidance and wisdom, which have contributed towards this thesis as well as any other success I have ever attained.

I would also like to express my thanks to several people who were closely involved with this thesis: Professor Luqi for her contributions in the foundation of this work and for her guidance during this research; Bernd Kraemer whose knowledge and patience I relied upon extensively; LCDR Yurchak and LCDR Griffin for their advice and encouragement.

Discussions with many of the faculty, staff and students in the Computer Science Department, too numerous to thank individually, contributed towards my understanding of this work and helped me to finish this thesis in the given time frame.

I. INTRODUCTION

This thesis describes the development of a rapid prototyping environment. This chapter presents a brief description of the software engineering problem and current methodologies utilized in software development. The Prototype System Description Language (PSDL), the Computer Aided Prototyping System (CAPS) and PSDL prototypes in CAPS are described briefly in the next chapter to provide a foundation for this thesis. The design of CAPS, and more detailed discussions of many of the primary tools in the environment are contained in successive chapters. These chapters are followed with our conclusions and recommendations for further research.

A. SOFTWARE DEVELOPMENT

The United States Department of Defense (DoD) is currently the world's largest user of computers. Each year billions of dollars are allocated for the development and maintenance of progressively more complex weapons and communications systems. These systems increasingly rely on requirements for systems which process information utilizing embedded computer systems. These systems are often characterized by maximum time periods or deadlines within which some event must occur. These are known as hard real-time constraints. Satellite control systems, missile guidance systems and communications networks are examples of embedded systems with hard real-time constraints. Correctness and reliability of these software systems is critical. Software

1

development of these systems is an immense task with increasingly high costs and potential for misdevelopment [1].

Over the past twenty years, the technological advances in computer hardware technology have reduced the hardware costs of a total system from 85 percent to about 15 percent. In the early 1970s studies showed that computer software alone comprised approximately 46 percent of the estimated total DoD computer costs. Of this cost, 56 percent was devoted specifically to embedded systems. In spite of the tremendous cost, most large software systems were characterized as not providing the functionality that was desired, took too long to build, cost too much time or space to use, and could not evolve to meet the user's changing needs [1].

Software engineering developed in response to the need to design, implement, test, install and maintain more efficiently and correctly larger and more complex software systems. The term *software engineering* was coined in 1967 by a NATO study group, and endorsed by the 1968 NATO Software Engineering Conference [2]. The conferees concluded that software engineering should use the philosophies and paradigms of traditional engineering disciplines. Numerous methodologies have been introduced to support software engineering. The two major approaches which underlie these different methodologies are the waterfall model [3] of development with its variants such as the spiral model [4], and the prototyping [5] method of development.

The waterfall model describes a sequential approach to software development as shown in Figure 1-1. The requirements are completely determined before the system is designed, implemented and tested. The cost of systems developed using this model is

very high. Required modifications which are realized late in the development of a system, such as during the testing phase, have a much greater impact on the cost of the system than they would have if they had been determined during the requirements analysis stage of the development. Requirements analysis may be considered the most critical stage of software development since this is when the system is defined [6].

| SYSTEM REQUIREMENTS |
| VALIDATION |

| SOFTWARE REQUIREMENTS |
| VALIDATION |

| PRELIMINARY DESIGN |
| VALIDATION |

| DETAILED DESIGN |
| VALIDATION |

| CODE AND DEBUG |
| DEVELOPMENT TEST |

| TEST AND PREOPERATIONS |
| VALIDATION TEST |

| OPERATIONS AND MAINTENANCE |
| REVALIDATION |
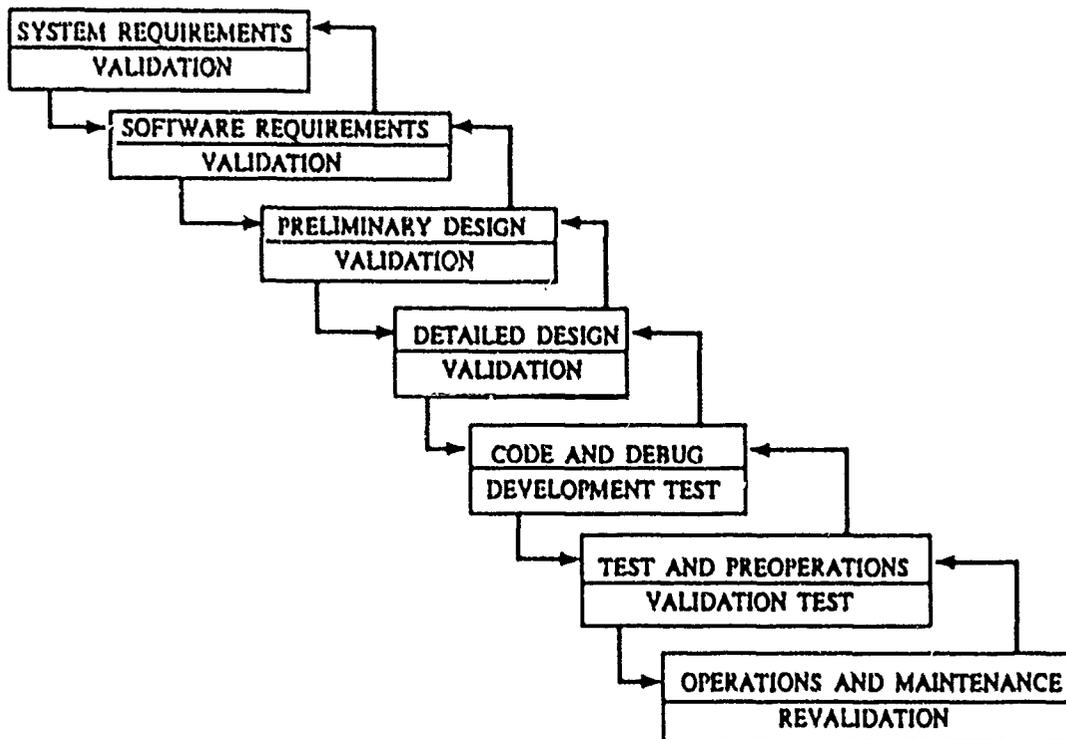
Figure 1-1. The Waterfall Model

Requirements are often incompletely or erroneously specified due to the often vast difference in the technical backgrounds of the user and the analyst. It is often the case that the user understands his application area but doesn't have the technical background to communicate successfully his needs to the analyst, while the analyst is not familiar enough with the application to detect a misunderstanding between himself and the user.

3

The successful development of a software system is strictly dependent upon this process. The analyst must understand the needs and desires of the user and the performance constraints of the intended software system in order to specify a complete and correct software system. Requirements specifications are still most widely written using the English language, which is an ambiguous and non-specific mode of communication.

B. RAPID PROTOTYPING

The waterfall model lacks automation support. Systematic support using computer aided tools has generally been unsuccessful for the waterfall model due to the informal and heuristic nature of software system design. Formal modeling of software systems and formal modeling of software development processes are key issues in automating the software design process. Systematic reuse of software or design knowledge has also been difficult because of the lack of specifications and explicitly recorded software design knowledge.

Prototyping captures selected aspects of the proposed system by generating executable models during the requirements analysis stage of software development. Trial use of these models and feedback from the users are major mechanisms used to determine if the defined system truly meets the user's needs before the system is designed, implemented and tested. When the requirements have been validated, the final version of the executable prototype provides a skeleton version of the critical aspects of the proposed software system. The prototype design should be extended into the production version of the proposed software system. This significantly reduces the cost and time of software development [5].

4

The rapid iterative construction of prototypes within a computer aided environment automates the prototyping method of software development and is called *rapid prototyping*. Computer support shortens the feedback cycle and introduces a degree of formality into the process of determining the requirements specifications. This offers a systematic way to rapidly turn the requirements specifications into executable prototypes which can be observed and tested in their natural environment. Reuse of system components in the process reduces the cost and effort in the iterative process. A rapid prototyping model [7] as applied to the requirements analysis phase of software development is shown in Figure 1-2.
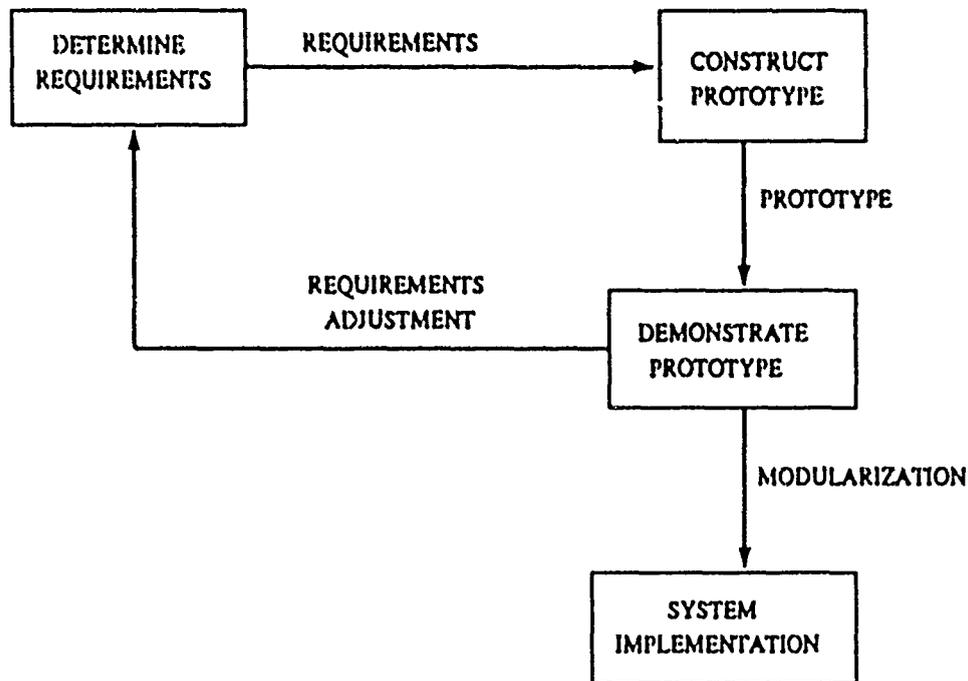
Figure 1-2. A Rapid Prototyping Model

A formal automated prototyping process requires a *rapid prototyping environment* [8]. Such an environment can provide the designer with an integrated set of tools which are used to design and test prototypes interactively. The requirements analyst and the user may both observe the behavior of the prototype and ensure that the requirements specifications meet the needs of the user. Rapid prototyping provides an efficient and precise means to determine the requirements for a software system, and greatly improves the likelihood that the software system developed from these requirements will be complete, correct and satisfactory to the user.

This thesis is part of a comprehensive framework for computer aided prototyping which includes language support, methodological support and tool support. The computer aided prototyping system is a pioneering effort with a long term impact on the automation of software design.

II. BACKGROUND

A. THE PROTOTYPING SYSTEM DESIGN LANGUAGE (PSDL)

PSDL [9] was designed as a prototyping language to provide the designer with a simple means to specify a high-level description of a software system. PSDL is an ideal language for a rapid prototyping environment and is the prototyping language used by the Computer Aided Prototyping System (CAPS). The design of PSDL places a strong emphasis on modularity, simplicity, reuse, adaptability, abstraction, and requirements tracing [5].

Modularity is essential for effective modification. Good modularity implies a prototype which is realized by a set of independent modules with narrow and explicitly specified interfaces. PSDL supports this concept by means of operators and data streams. Two distinct operators can communicate or affect each other's behavior only when a data stream explicitly connects the two operators.

Simplicity is supported by the small set of powerful constructs provided in PSDL. PSDL designs are networks of operators connected by data streams. These networks can be represented as data-flow diagrams augmented with timing and control constraints. Operators in the system can represent functions or state machines. The data streams carry exception conditions or values of arbitrary abstract data types.

PSDL supports reuse through uniform specifications suitable for retrieving reusable components from a software base. The specification part of a PSDL component contains

7

several attributes which describe the interface and behavior of the component. These attributes can be used to generate automatically uniform specifications for storing and retrieving reusable components.

PSDL supports adaptability through its ability to make small modifications to modules by means of the control constraints. Control constraints affect modules in several ways. They can be used to impose preconditions on the execution of a module, filter the output of a module, suppress or raise exceptions in specified conditions, and to control timers. These facilities provide the means to modify the behavior of a module independently of its implementation.

PSDL provides abstractions suitable for describing large systems which may contain real-time constraints. These abstractions include control constraints, timing constraints, timers, functional abstractions and data abstractions.

PSDL supports requirements tracing by means of a construct for declaring the requirements which are associated with each part of a prototype. This is important because the prototype must adapt to the changing perceptions of the requirements resulting from evaluations of the prototype behavior.

The computational model underlying PSDL can be described by an augmented graph $G = (V,E,T,C)$. V is a set of vertices. E is a set of edges. $T:V->R \cup \{ \infty \}$ (where R denotes the set of real numbers) is a function assigning the maximum execution time for each vertex in V. PSDL permits bounded and unbounded maximum execution times. $C:V->R \cup \{ \infty \}$ is a function assigning the control constraints for each vertex in V. In this graph a vertex represents an operator and an edge is a data stream.
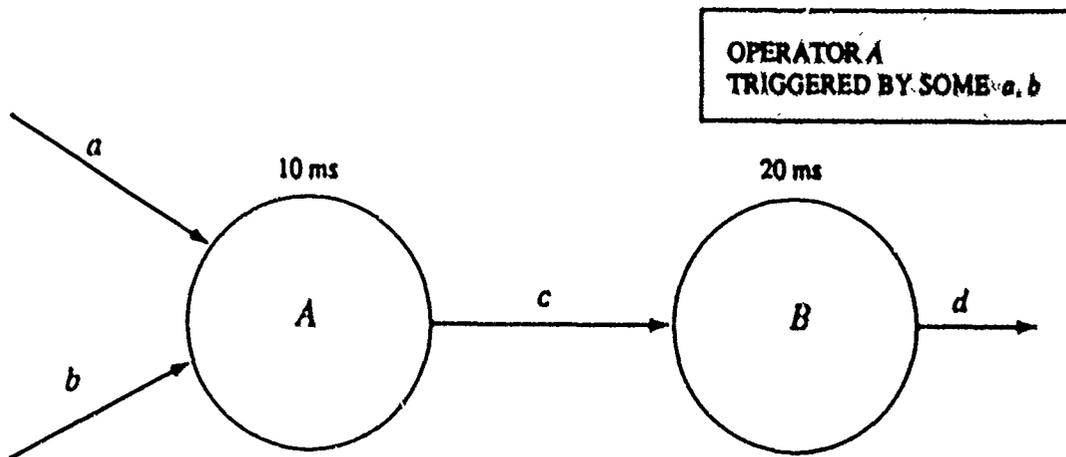
Figure 2-1. PSDL Graph

Figure 2-1 shows an example of a PSDL design graph with operators A and B, and data streams a, b, c, d. The graph also indicates timing constraints, 10 ms for A and 20 ms for B. Control constraints are provided for operator A. The intended meaning of this specification is that operator A receives input data on data streams a and b, processes the data within 10 ms, and outputs data on data stream c. Operator B receives input data on data stream c, processes the data within 20 ms, and outputs data on data stream d.

Operators represent functions or state machines. A function produces output whose value is solely dependent upon the input values. A state machine produces output whose value depends upon the input values and on internal state values representing some part of the history of computation. Operators can be triggered either by the arrival of input data values or by periodic timing constraints which specify the time intervals for which an operator must fire. PSDL operators are atomic or composite. Atomic operators represent single operations and cannot be decomposed into subcomponents. Composite

9

operators represent networks of operators and data streams into which the operators may be decomposed. Operators are also either periodic or sporadic. Periodic operators fire at regular intervals of time while sporadic operators fire when there is new data on a set of input data streams.

Data streams represent sequential data flow mechanisms which move data between operators. Data streams are either data flow data streams or sampled data streams. Data flow data streams are similar to FIFO queues with a length of one. Any value placed into the queue must be read by another operator before any other data value may be placed into the queue. Values read from the queue are removed from the queue. Sampled data streams may be considered as a single cell which may be written to or read from at any time and as often as desired.

Timing constraints are essential for real-time systems. The timing constraints impose an order on operator firing which is based on timing rather than on data flow. There are three basic types of timing constraints: 1) maximum execution time, 2) deadline or maximum response time, and 3) minimum calling period. Maximum execution time is an upper bound on the length of time that an operator may use to complete its function. Deadlines apply only to periodic operators and maximum response times apply only to sporadic operators. For periodic operators, the deadline is an upper bound on the time between the beginning of a period and the time that the operator places the last output value onto a data stream during a period. For sporadic operators, the maximum response time is an upper bound on the length of time between the arrival of one or more new data values on an input data stream and the time when the final output is placed on an output data stream. The minimum calling period applies only to sporadic operators

10

and represents a lower bound on the time between the arrival of one set of inputs and the arrival of another set of inputs.

Control constraints are the mechanisms which refine and adapt the behavior of PSDL operators. They specify how an operator may be fired, how exceptions may be raised, and how or when data may be placed onto an operator's output data streams.

The PSDL Grammar is shown in Appendix A.

B. THE COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

CAPS is a unique rapid prototyping environment which includes the ability to prototype hard real-time systems. CAPS utilizes PSDL and an integrated set of prototyping tools. The tools are integrated through the user interface. The primary tools in CAPS may be divided into three main subsystems [5]. The subsystems and their tools are:

(1) the User Interface which is comprised of:
   - a Graphic Editor [10]
   - a Syntax Directed Editor [11]
   - a Browser [5]
   - an Expert System [5]

(2) the Software Database System which is comprised of:
   - a Software Design Management System [5]
   - a Design Database [12]
   - a Software Base [13]

(3) the Execution Support System which is comprised of:
   - a Translator [14]
   - a Static Scheduler [15,16]
   - a Dynamic Scheduler [17]
   - a Debugger [17]

The Graphic Editor is a tool which permits a designer to specify the portions of a PSDL prototype using graphical objects to represent the system. Graphical objects include operators, inputs, outputs, data flows and self loops on operators. All graphic objects are named and may have time constraints associated with them.

The Syntax Directed Editor is used by the designer to enter the textual portions of the prototype design not represented by the graphic editor and to ensure that the prototype is syntactically correct PSDL.

The Browser provides a means for the designer to view reusable components in the software base.

The Expert System provides a paraphrasing capability that generates English text descriptions of PSDL specifications. This tools permits users who are unfamiliar with the PSDL language to evaluate a prototype.

The Software Design Management System manages and retrieves the versions, refinements and alternatives of the prototypes in the design database and the reusable components in the software base.

The Design Database contains PSDL prototype descriptions for all software projects developed using CAPS.

The Software Base contains PSDL descriptions and implementations for all reusable software components developed using CAPS.

The Translator generates high level code from the PSDL prototype which binds the reusable components from the software base to the executable prototype.

The Static Scheduler attempts to allocate time slots for the representations of PSDL operators with real-time constraints before the prototype is executed. If the allocation succeeds, all operators are guaranteed to meet their deadlines.

The Dynamic Scheduler invokes representations of operators without real-time constraints at run-time to occupy time slots which are not used by operators with real-time constraints. The time slots which the dynamic scheduler uses are considered as "slack times". Dynamic scheduling occurs during execution of the prototype.

The Debugger allows the designer to interact with the execution support system. The debugger has facilities for initiating the execution of a prototype, displaying execution results or tracing information of the execution, and gathering statistics about a prototype's behavior and performance.

Prior to the work described in this thesis, partial implementations had been developed for the graphic editor, translator and the static scheduler. Designs of an expert user interface and a debugger had been defined. Feasibility studies had been conducted for the syntax directed editor, design database and the software base.

C. PSDL PROTOTYPES IN CAPS

PSDL prototypes are described by the designer in the graphic editor or the syntax directed editor. Once a prototype has been specified, the tools within the execution support system will construct an executable view of the prototype and then actually execute the prototype. The execution support system assumes a syntactically correct PSDL description.

PSDL prototype components are either an operator or a data type. A PSDL prototype may contain multiple type and/or operator components. All PSDL components have a specification part and an implementation part. The implementation part may either be a PSDL implementation or an Ada implementation.

The purpose of the two different implementation constructs is to provide a simple means for decomposing a prototype. A PSDL prototype can be represented by a tree structure. The leaves of the tree are atomic level components and they contain Ada implementation parts. Figure 2-2 contains a simple top level graphic representation of a PSDL prototype.
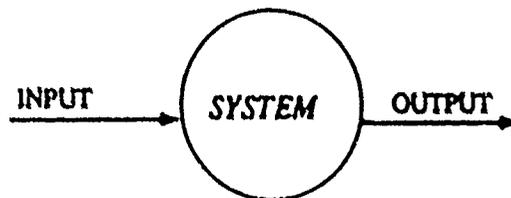
Figure 2-2. PSDL Prototype

If the node *SYSTEM* has a match in the software base of reusable Ada software components, then the node *SYSTEM* is atomic and comprises a complete description of the prototype *SYSTEM*. In this case no further decomposition is required. If a match for *SYSTEM* is not found in the software base then the node *SYSTEM* is considered a composite operator. In this case the designer has two choices. If the designer does not recognize a conceptual decomposition of *SYSTEM* then the designer considers *SYSTEM* as atomic although an Ada representation does not already exist. The designer may then provide an Ada implementation for *SYSTEM*. The Ada implementation will be internally

14

substituted for the PSDL implementation. The prototype would then be completely described and is atomic. The Ada implementation will become a persistent component in the software base and will be matched in future sessions with CAPS. If the designer realizes a decomposition for *SYSTEM*, then the designer will decompose the composite operator with *SYSTEM* as the root of the tree. An example of a possible graphic decomposition is shown in Figure 2-3. The new operators resulting from the decomposition are children of the operator *SYSTEM* in the tree structure.
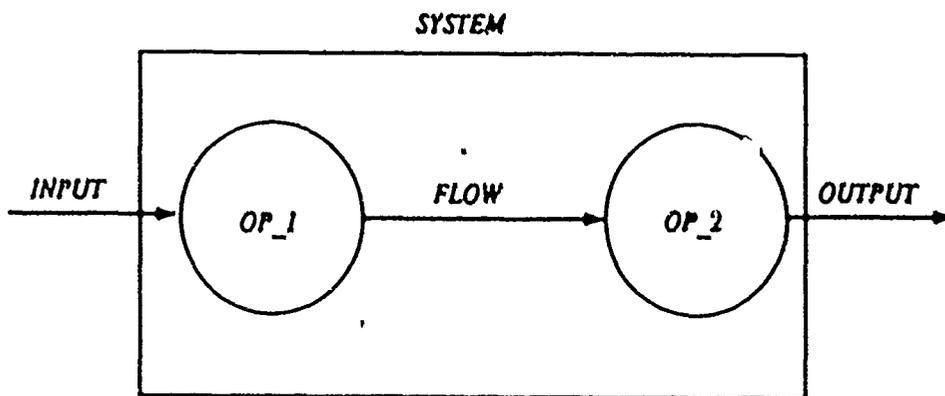
Figure 2-3. Decomposed PSDL Prototype.

The identification and decomposition is recursively applied until the leaves of the tree all contain Ada implementations. A possible complete tree structure for *SYSTEM* can be a single node. The complete tree structure will have a depth which is defined by the maturity and modularity of the software base and/or the designer's conceptual model of a properly decomposed system. A possible complete tree structure for a PSDL prototype is shown in Figure 2-4. It has two composite operators; SYSTEM and OP_2, and three atomic operators OP_1, OP_2A and OP_2B.
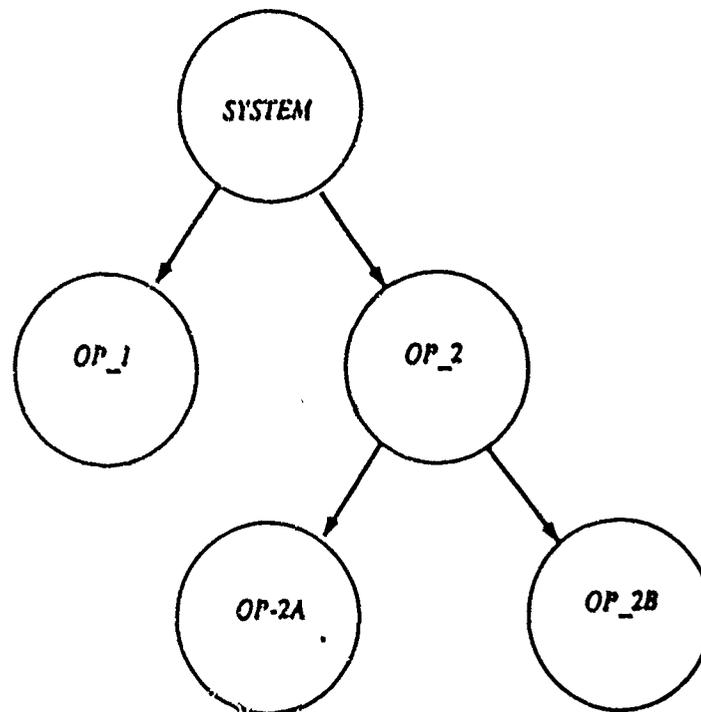
Figure 2-4. Tree Structure For PSDL Prototype.

A particular software base will eventually contain decompositions which reflect the style of thinking used by its designers. It will become customized to its users as it matures. The reusable components which are added to an initial software base reflect the level of decomposition that the designers utilize in describing the systems which they decompose. If the designers decide to provide Ada implementations at a high level of decomposition then the reusable components may tend to be large in size and may generally be very specific to certain applications. If the designers decompose the system into very small modules, then the Ada implementations in the reusable software base will tend to be very small and will more likely occur more frequently in the design of successive systems.

16

The evolution of the software base of reusable components reflects the designers preference between using PSDL and using Ada to express their own concepts of a system. The design database will also reflect the style of its users.

The success of any rapid prototyping environment is strongly linked to the means by which the designers may describe their prototypes. CAPS provides two means of describing prototypes: the graphic editor and the syntax directed editor. The majority of the designs in the design database will eventually be in the form that the users prefer.

The provision of two means for user input is an important human factors feature. CAPS does not attempt to constrain a designer to one mode of input. While visual programming [18] is gaining momentum in research, the designers of CAPS recognize that constraining a user to one particular mode of input which emphasizes just one method of abstraction might possibly reduce the effectiveness of the environment when applied across a broad sample of users.

A current research question with regard to rapid prototyping is whether or not it is really necessary to design a new language specifically for prototyping, and if so, what are the necessary features and characteristics of the language?

The influence of any language on the problem domain must be considered. In this regard, MacLennan refers us to the Sapir-Whorf hypothesis [19] which states that the structure of language defines the boundaries of thought. The use of a given language does not prevent certain thoughts but does facilitate or impede certain modes of thought. This lends support to the idea that a careful choice of language can reduce the conceptual barriers in the design of embedded software systems.

17

Computer languages have evolved over time in response to the perceived problems of existing languages. As languages have evolved, important principles which relate to the design, evaluation, and implementation of languages have evolved as well. Ada has been designed as a response to DoD's need for a standard language which provides all of the features we currently perceive as necessary to develop and maintain large software systems. Ada contains features which were not encompassed in any one previously existing language. Since our rapid prototyping environment is aimed at large, real-time, embedded systems, our executable prototypes must be able to be expressed in Ada. Ada is a complex language which supports abstraction, information hiding, modularity, localization, uniformity, completeness and confirmability. The complexity of Ada leads to the need for a prototyping language.

The prototyping language must provide a simple yet expressive means for a designer to describe a system. The prototyping language must support a cost effective means to establish the requirements of a system. The prototyping language must support the features of the Ada programming language. PSDL meets these requirements [20].

III. DESIGN ISSUES FOR THE DEVELOPMENT OF CAPS

The design and feasibility study of CAPS has been partially influenced by local resources, personnel and equipment. Although the primary researchers are permanently assigned, the secondary researchers are mostly students with diverse backgrounds who are able to devote only a short period of time towards development. Student contributions have varied in methodology and programming languages used, and they reflect the somewhat diverse backgrounds of the many students who have been involved. The development of CAPS has been a long term project. The design and feasibility study for the implementation of the underlying prototyping language has occurred over the last five years, and continues to progress.

A. CAPS SYSTEM CONFIGURATION

The initial step in the development of our rapid prototyping environment was to define an interface between tools and a system configuration. The best utilization of previous work in the design and implementation of CAPS tools was an essential factor in our decision. The previous development of CAPS tools utilized and assumed the availability of a Sun Workstation.

There were two primary methods of integration considered for the initial development of CAPS. One method was to define an interface which would manage a collection of loosely coupled tools. The second method was to define a software architecture as a foundation for tool integration.

The Sun Operating System provides the simple UNIX [21] interface of the text stream. The UNIX method of tool integration defines an environment which consists of a collection of loosely coupled tools. An advantage of this method is its simplicity. An environment shell program which provides the interface with the user and manages the communications between the tools provides an easy means to integrate new tools and to extend the capabilities of existing tools. A disadvantage of this method of integration is that it may result in multiple data components which represent a prototype. Each data component produced by a particular tool is most likely a partial view of a complete prototype which reflects the transformation of data performed by each particular tool. Other disadvantages of this method are that the interface must be modified to add or remove tools from the system and that the environment contains a lesser degree of integration granularity.

An example of the second method of integration is the Illinois Software Engineering Program (ISEP) [22] which describes an open systems architecture for tool integration. This method defines a software bus which provides for the interconnection and intercommunication between the tools within the environment. All tools communicate with other tools by means of a set of communication protocols specially defined for each tool. Any two tools may communicate which have a common set of protocols. The advantage of this methodology is that it is intended to support the integration of new tools independently of the other tools in the environment. The disadvantage of this methodology is that it requires the development of a complex set of protocols for each tool in the environment.

The previous work on the bottom-up development of the CAPS tools assumed the first method of integration. Due to this assumption and its simplicity we chose to connect the tools within the CAPS environment in the UNIX fashion by passing streams of data between the different tools.

The system configuration for CAPS is a natural decomposition of the CAPS design. A *system directory* is considered the root of the environment with subdirectories for each of the tools. The subdirectories contain the required components that make up each of the tools and contain documentation specific to each tool.

B. METHODOLOGY FOR INTEGRATION

The first implementation requirement for the environment is the user interface, since it is the user interface which manages the tools in the environment. Once a basic user interface was designed, we realized that the process of integration would be a somewhat circular process with iterative refinements.

The interdependence of the CAPS tools required us to simulate the functionality of some tools while testing other tools. We decided that the most natural tools to include in the initial integration process were the tools within the user interface. The directory *prototypes* in the system configuration simulates the storage locations of both the design database and the software base. We realized that the database management functions could be performed manually until development, integration, and testing of the databases could be performed.

A systematic approach for the integration of a partially developed set of tools was established which would make the best use of previous work. The integration methodology for the tools consists of ten primary steps:

Step 1:

Identify, locate and relocate all of the subcomponents and products of a particular tool. The subcomponents were relocated to form an identifiable modular representation of each particular tool.

Step 2:

Determine the dependencies between the subcomponents of each tool and to determine how each subcomponent was intended to interact with the other subcomponents.

Step 3:

Resolve any naming conflicts which existed with the subcomponents of a tool or the data it produced with the names expected by other tools in the environment.

Step 4:

Determine how to compile the various subcomponents, and to determine which system libraries needed to be linked with each subcomponents.

Step 5:

Separate the functional components of a tool from the partial views of a prototype which were used by the previous developers to test their tools and to load these subcomponents in the proper places within our system configuration.

Step 6:

Test each tool to validate the functionality of the tool with its previous documentation. In most cases the previous documentation did not make much distinction between the long term designs and the state of the actual implementations of the tools.

Step 7:

Identify areas of functionality which would significantly improve the usefulness of each tool.

Step 8:

Correct bugs discovered during the sixth step, implement features of the tools which were described in the previous design but which had not actually been implemented; and implement features identified in step seven.

Step 9:

Identify test cases which were appropriate for the current state of the environment and test the tool from within CAPS.

Step 10:

Document the results of the ninth step and to document any new ideas which resulted from the integration testing and evaluation of the current tool.

C. PORTABILITY AND SYSTEM DEPENDENCIES

The primary goal of the CAPS designers is to focus on the design and feasibility of developing a rapid prototyping environment and not on the development of a production

23

system. An early decision was made to accept dependence upon the best locally available resources. Portability of our particular environment is considered a secondary goal.

Our implementation currently uses the local Sun system configuration and depends on features of the local installation such as the server and printer names on our Sun NFS. The use of these local resources enhances the development process by permitting more flexibility in the use of local resources shared by many other students and faculty at our research location. The use of system dependent name definitions have been localized to the front end of the user interface. This localization allows the system to be redefined with minimum effort.

An early decision was also made to utilize existing tools for the development of our own tools and our environment. Part of the research involved with the development of our environment and its tools was to survey and evaluate existing tools which were potential candidates for use in developing our tools or as candidates to be integrated into CAPS to represent one of our tools.

IV. THE USER INTERFACE

A. PREVIOUS DESIGN

The previous design of an expert user interface [23] defines an interface which provides sequence control with data protection. The user interface guides a user through the rapid prototyping process according to the following guidelines [23]:

- "The interface must be able to interpret what the user is doing at any time and provide support".

- "The expert system must communicate with the users to find out what they want to do at any moment when the system cannot be sure of the user's intentions".

The following goals were defined in the design of the user interface:

- Required input data should only be entered once.
- Feedback should always be provided during data entry.
- The user interface should be adaptable to accommodate both the novice and the experienced user.

The major commands defined in the top level user's manual were:

- caps
- construct
- execute
- modify

B. PREVIOUS IMPLEMENTATION

An implementation for the user interface had not been previously developed. Raum [23] described and implemented a link statement analyzer as part of the user interface.

The link statement analyzer translates the link statements generated by the graphic editor into textual PSDL constructs.

The input to the link statement analyzer is a data component produced by the graphic editor and stored in the file *graph.links*. The data in this file are PSDL link statements which represent the graphic structure of a PSDL design. The form of a link statement without an optional maximum execution time is *name.source->sink* where *name* is the name of the data, and *source* and *sink* are either the keyword EXTERNAL or names of operators. The form of a link statement with an MET is *name.source:MET->sink*. Figure 4-1 (a) shows a PSDL design produced with the graphic editor. The corresponding link statements generated are depicted in Figure 4-1 (b).

The link statement analyzer processes the link statements and generates two new data components. A file called *psdl.ds* is created which contains a list of all PSDL data streams in the graph. Additional files are created for each operator in the graph. These files are called *NewNode.XX*, where the *XX* represents arbitrarily assigned consecutive numbers. The *NewNode.XX* files are textual PSDL specification parts which contain all the information about an operator which was entered through the graphic editor. These files are intended to represent the prototype in the design database. This design does not account for multiple instances of prototypes, or describe how related components are linked together in the design database. Figure 4-2 (a) shows the data component *psdl.ds* and Figure 4-2 (b) shows the data components *NewNode.01* and *NewNode.02*.

Although the link statement analyzer was first designed as part of the user interface, we now consider it to be a component of the graphic editor.

26
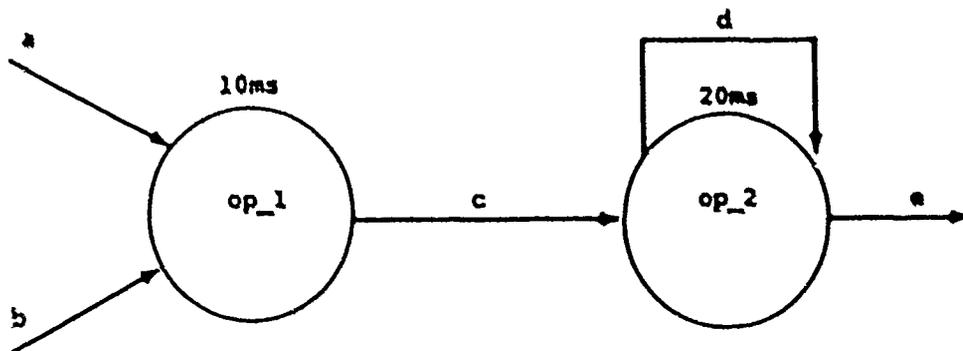
(a)

```
a.EXTERNAL->op_1
b.EXTERNAL->op_1
c.op_1:10 ms->op_2
d.op_2:20 ms->op_2
e.op_2:20 ms->EXTERNAL
```

(b)

Figure 4-1. PSDL Graph and Link Statements

```
DATA STREAM c
```

(a)

```
OPERATOR op_1
SPECIFICATION
   INPUT   a
           b
   OUTPUT  c
   MAXIMUM EXECUTION TIME 10 ms
END
```

```
OPERATOR op_2
SPECIFICATION
   INPUT   c
   STATE   d
   OUTPUT  e
   MAXIMUM EXECUTION TIME 20 ms
END
```
(b)

Figure 4-2. Products Of The Link Statement Analyzer

C. MODIFICATIONS TO THE DESIGN

Our modifications and enhancements to the previous design addressed the following

issues:

- the responsibilities of the user interface
- a methodology for user interaction
- the menu functions
- view consistency

1. User Interface Responsibilities

We redefined the meaning of the user interface for implementation purposes. Raum's design of the user interface maintained the idea that the user interface was one of the three main components of CAPS. Raum described Bourne Shell Scripts which only managed the activities of the components of the user interface, such as the graphic editor, syntax directed editor, browser, expert system and the debugger. As described in the previous chapter we now view the user interface as the shell of the environment which interacts with the user and manages all of the tools within the CAPS environment and not just the tools which interact with the user. The effect of this modification changes the responsibilities of the user interface.

The *caps* command was originally intended to be used to place the user into the user interface portion of CAPS. The original *caps* command was to allow an optional argument to assign a name to a new prototype. The *caps* command is now used to enter the CAPS environment. The specification of a prototype name is not appropriate at this level and the names of prototypes will be controlled from within the environment.

2. Methodology For User Interaction

We decided to design a simple menu driven interface for our initial integration. Since the functionality of the user interface had not yet been well defined, we wanted to provide a simple mechanism for integrating the previous work on the various tool. We chose text menus as our first mechanism for user interaction. We decided to utilize mnemonic lettering for option selections. Selections are made by typing the first lower case letter of an option at the prompt. The character selection mechanism only requires

29

that the user remember the activity that he wishes to be performed. This approach supports type-ahead selections and system evolution [24]. Ultimately, we intend to implement a graphic interface.

3. Menu Functions

We defined menus which guide the user through the process of rapid prototyping. The main menu contains the functions:

- construct
- execute
- modify
- quit

The *construct* option places the designer in another menu which displays the choices of construction tools. Currently the choices are to use the graphic editor or the syntax directed editor.

The *execute* option activates the execution support tools. These tools are the translator, the static scheduler and the dynamic scheduler. After these tools produce the data components which represent the executable prototype, the data components are automatically compiled, linked, and executed.

The execution status messages were previously defined as:

Translation Complete
Static Scheduler Complete
Dynamic Scheduler Complete
Compilation Complete
Linking Complete
Execution Complete

These messages were used to bridge the wait time the user experienced while these actions are performed. These messages were to be displayed after the action occurred.

We realized that a message which informs a user of the current activity rather than the previous cause of delay, is generally more satisfactory. The status message for linking was determined to be unnecessary. The execution support tools each produce data components which partially represent the executable prototype. All data components are compiled and linked during the compilation state. We defined two status messages which reflect the execution of the prototype. Since the execution of an embedded system implies a continuous system, we defined an additional status message which responds to a user input of turning off the system. The current status messages are:

Translating ...
Building Static Schedule ...
Building Dynamic Schedule ...
Compiling ...
Executing ...
Execution Complete

The *modify* option had not yet been well defined in the previous documentation. We have defined the *modify* process to be equivalent to the *construct* process with one major difference. When the *modify* option is selected a window should be opened which contains the top-level names of all existing prototypes in the design database. The user should be able to select a prototype for modification by selecting an entry with the mouse. If the user selects a prototype with a *.graph* suffix, the graphic editor should be automatically activated by the user interface. If the user selects a prototype with a *.text* suffix, the syntax directed editor should be automatically activated by the user interface.

The *quit* option was not defined in the previous documentation. The *quit* option should be used to permit the user to clean up versions of prototype designs which were

31

generated during the current session before exiting the environment. This process could occur as shown in Figure 4-3.

```
Do you wish to save:

        prototype1.text?   (y/n)  :  y
        prototype1.graph?  (y/n)  :  y
        prototype2.text?   (y/n)  :  n
        prototype2.graph?  (y/n)  :  n
```

Figure 4-3. Quit Process

---

4. View Consistency

A view is a representation for a particular abstraction. Multiple views may be associated with the same abstraction. View consistency defines a principle where multiple views of the same abstraction are always equivalent. This means that a change in one view must be reflected in all other views which represent the same abstraction. View consistency between the graphic and textual representations of prototype designs was an important issue of integration which still requires further consideration.

Since the construction of a prototype design is expected to be an iterative process, consistency issues are an immediate concern. Because CAPS supports the evolution of software systems which may be developed and maintained by a large number of developers, we should not constrain the view of a prototype to the preferences of one particular user. A view of a prototype may be required by multiple users in the development process, and may be required by many other users during evolution.

32

The view consistency problem is affected by the capabilities of the graphic editor and the syntax directed editor. This means that the problem is different dependent upon whether a prototype may be completely or partially described within each tool. If we constrain the capability of the graphic editor to describe only a simple data-flow diagram which represents the PSDL *graph* construct and not permit the capability to completely describe the prototype, then we define this as a partial graphic view. If the syntax directed editor does not permit the capability to describe textually the link statements of a PSDL *graph* construct, then we define this as a partial textual view. If a prototype may be described completely within the graphic editor or the syntax directed editor, then we define these views as a complete graphic view or a complete textual view. The method of interaction between the construction tools will be dependent upon the final capabilities of the tools.

An initial view consistency exists for prototypes designed with the graphic editor. The graphic editor generates a textual representation which is used by other tools later in the rapid prototyping process. Our initial pipeline communication design will not adequately support view consistency for prototypes which were initially described as a textual representation or for modified textual representations.

The problem with view consistency for textual representations is that the two dimensional translation of logical objects into a graphic representation does not have a generally satisfactory solution. Even if both tools utilize a common data structure, the logical objects in the common data structure would still require a graphic translation for the layout used in the graphic editor.

33

One solution which supports view consistency is to define the graphic editor as the primary construction tool from which the prototype is completely described. The interaction with the graphic editor would produce a partial graphic view and interaction with the syntax directed editor would produce a partial textual view. The graphic editor would utilize the syntax directed editor as an underlying mechanism to provide for describing its textual attributes. The interaction between the editors must be defined so that objects which are defined graphically cannot be modified by the syntax directed editor. This requires that the user have limitied access to the textual representation of a prototype within the syntax directed editor.

C. IMPLEMENTATION

The user interface was implemented in the C programming language [25]. This language was chosen due to the ease with which it interfaces with the unix shell. The use of the C programming language gave us the power and structure of a high level language and still provides very easy access to shell commands from within the program to manage the various CAPS tools.

The simplicity of our menu design enables a user to traverse the system with an ease comparable to that provided by graphic interfaces. If an invalid selection is entered by the user, an error message is displayed, and the user may enter another selection. An sample of the menu design is provided with the main menu in Figure 4-4.

```
COMPUTER AIDED PROTOTYPING SYSTEM

        (c) onstruct
        (e) xecute
        (m) odify
        (q) uit

Select Option:
```

Figure 4-4. CAPS Main Menu

Each valid selection from a menu clears the screen, then either places the user in a submenu if further choices are available or places the user within the environment of a particular tool. This occurs by creating a new unix process which executes the desired tool as a concurrent process with the user interface or any other currently active tools. The implementation of the user interface is contained in Appendix B.

V. THE GRAPHIC EDITOR

A. PREVIOUS DESIGN

The graphic editor [10] was designed to support efficient construction and modification of the graphical representation of PSDL prototypes. The graphic editor assumes that it is running on a Sun Workstation with a three button mouse. It uses both keyboard and mouse inputs. The control options of the graphic editor include *load existing* prototype, *store* current prototype, and *quit* the graphic editor.

Graphic representations of PSDL prototypes can be created by selecting the following editing modes:

- draw operator
- draw input
- draw output
- draw data stream
- draw self loop

The graphic symbols which represent the corresponding language constructs are created by the following process:

1. Position the mouse locator at the desired position in the drawing space.

2. Press the left mouse button down.

3. While holding the left mouse button down, move the mouse
   to a position which defines the size or length of the object.
   This rubber bands the type of object chosen in the editing mode.

4. Release the left mouse button when the desired length or location
   is obtained.

The keyboard and the text input mode are used to define identifier names and maximum execution times (MET). All operators, inputs, outputs data streams and self loops must have identifier names specified. Additionally all operators must additionally have METs specified. Objects are deleted by positioning the mouse on an object and pressing the right mouse button. Error messages are overlaid in the drawing space.

The layout of the previous graphic editor is shown in Figure 5-1.
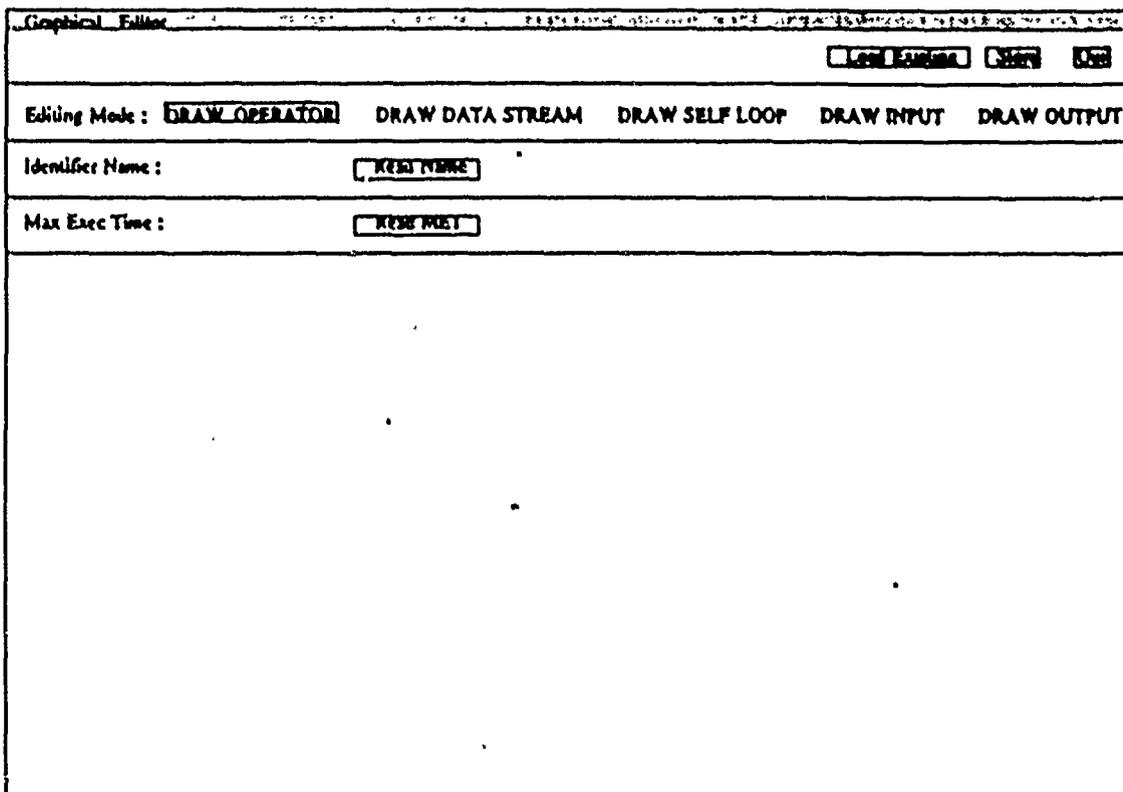
Figure 5-1. Previous Graphic Editor

All graphic representations of designs are to be stored and managed by the design database. Thorstenson [10] states that:

> "If the graphical editor is going to be used to edit an existing diagram, the user interface function must retrieve the necessary reconstruction information from the design database and store the information in a file named graph.pic prior to invoking the editor. The graphical editor then reads in this information and reconstructs the diagram."

The graphic editor only allows objects to be related to other objects as defined by PSDL. Operators are represented by bubbles. Bubbles may not overlap. Data streams are represented by arrows where both the tail and head of the arrows are connected to operators. Inputs are represented by arrows whose tails must be positioned in unoccupied drawing space and heads must be connected to an operator. Outputs are represented by arrows whose tails are connected to an operator and heads are positioned in unoccupied drawing space. Self loops which represent state variables appear as arrows whose tails and heads are both connected to the same operator. PSDL operator decomposition also requires that all of the components of an operator are named. The graphic editor will not permit an object to be entered into the drawing space until after the identifier name has been entered and validated in the text input mode. PSDL operators also have a MET associated with them which must also be specified before they may be drawn in the drawing space. The identifiers must be syntactically correct Ada identifiers. The form of a syntactically correct identifier is shown in Figure 5-2.

```
identifier       ::= letter ([underline] letter_or_digit)
letter           ::= upper_case_letter | lower_case_letter
letter_or_digit  ::= letter | digit
```

Figure 5-2. PSDL Identifier

The form of a syntactically correct MET is shown in Figure 5-3. User input which fails to meet the syntax requirements of PSDL is ignored, such as an input without an associated operator, or an output without an associated operator.

```
MET          ::= digit_string (time_unit)
digit_string ::= digit | digit_string
time_unit    ::= 'h' | 'm' | 's' | 'ms'
```

Figure 5-3. PSDL MET

Previous documentation did not describe any direct interface between the graphic editor and the syntax directed editor, nor did it describe the decomposition of a prototype design from within the graphic editor.

B. PREVIOUS IMPLEMENTATION

The implementation of the graphic editor utilizes the suntools, sunwindows, pixrect and math libraries on a sun 3 system. The implementation of the graphic editor consists of programs written in C, Pascal and UNIX C Shell.

C. INTEGRATION

The integration of the graphic editor into CAPS required that the component parts which comprise the graphic editor and the data components produced by the graphic editor be identified and relocated into our system configuration. The dependencies between the graphic editor components and the data components were also identified. The user interface was modified to activate the graphic editor when the designer selected it from within the construction menu.

The components which comprise the graphic editor reside in the directory: /caps/graphic_editor. These components are:

```
editor.icon    - icon for graphic_editor window
ge             - C shell script program
graph          - executable window based graphic editor
graph.c        - C source code for graph
makid          - executable utility to compile graph.c
makid.c        - C source code for makid
nodes          - graphic design link analyzer
nodes.p        - Pascal source code for nodes
```

The products of the graphic editor which represent views of a prototype are placed in the directory: /caps/prototypes.

The graphic editor is selected by the user interface (/caps/caps). The user interface calls the graphic editor by creating a new process, which is a copy of the current process, and then overlaying the new process with the script program, *ge*. This code segment is shown in Figure 5-4.

40

```
if (fork() -- 0) {
    code - execl(SHELL, SHELL, "-f", GRAPHIC_EDITOR, 0)
    exit(code)
}


SHELL          represents  /bin/csh
GRAPHIC_EDITOR represents  /n/suns2/work/caps/graphic_editor/ge
```

Figure 5-4. Code Segment

ge is dependent upon:

  /caps/graphic_editor/graph
  /caps/graphic_editor/nodes

ge creates the following file:

  /caps/prototypes/psdl.imp

graph.c is compiled by "makid graph.c"
which is equivalent to "cc graph.c -o graph -lm -lsuntool -lsunwindow -lpixrect"

  graph is dependent upon:
    editor.icon

  graph creates the following files:
    /caps/prototypes/graph.links
    /caps/prototypes/graph.pic

nodes.p is compiled by "pc nodes.p -o nodes"

  nodes is dependent upon:
    /caps/prototypes/graph.links

  nodes creates the following files:
    /caps/prototypes/psdl.ds
    /caps/prototypes/NewNode.XX's

41

D. INTEGRATION TESTING

The following inconsistencies, missing features or important enhancements were identified during the testing of the graphic editor.

1) The frame title in the previous documentation was stated as **Graphical Editor**. The frame title in the implementation was **dataflow diagram editor**.

2) There was no facility to print a hardcopy of a prototype design. The ability to print a hardcopy of the graphic representation of a prototype was also determined to be an important enhancement for documentation purposes. The use and capabilities of the graphic editor could be described in written documentation much more clearly with the ability to show the format of the graphic editor.

3) The graphic editor utilized only a portion of the monitor. The default window size of the graphic editor was proportional to the monitor and used about 75 percent of the available screen space.

4) A segmentation violation occurred intermittently. On a UNIX system a segmentation violation indicates that a pointer has an assigned address outside of the user's data space.

5) Whenever an input, output or data-flow was erased from the drawing space, the last pixel on the tail of an input, output or data-flow remained visible.

6) The functions of the mouse buttons were not visibly described within the screen image of the graphic editor.

7) The graphic editor warning and error messages were displayed in the drawing space, permitting the possibility of overlaying the graph.

8) The graphic editor has separate store and quit buttons yet the user was not permitted to exit the graphic editor via the quit button unless the design had been stored first.

9) The operator data components produced by the link statement analyzer were not complete specification constructs.

10) The command line help facility was not effective once the graphic editor was integrated into the environment.

11) The delete operation was unpredictable. This feature was not completely implemented in the previous implementation.

12) Inputs, Outputs and Data-flows were not clipped. These symbols are drawn exactly as specified by the mouse inputs and not adjusted to the edges of the operators. The impact of this is that the construction of a nice looking prototype can become tedious. This problem becomes worse on workstations with smaller monitors.

13) The graphic editor requires that the designer selects the tail position of an input, output, or data-flow before the designer selects the head position of the input, output, or data-flow. If the designer selects the end points of an input, output or data-flow in the head then tail order, the input, output or data-flow simply disappears without any explanation as to why the object was ignored. This is a cumbersome and unnecessary constraint on the user.

14) The objects in the graphic editor cannot be resized.

15) The objects in the graphic editor cannot be moved once positioned.

16) Data flows cannot be inverted.

17) The keyboard inputs for the identifier names and maximum execution times are translated to literal text characters. Backspaces used while entering these fields appear to behave normally but show up as control characters in the output files.

18) The names of objects in the graph are not required to be unique.

19) Names and time constraints may not be modified once their associated object is placed in the drawing space.

20) Units for time constraints in the PSDL language definition have a default value of milliseconds. The graphic editor does not recognize this feature of the language.

21) The previous design assumed that only one single graphic view of a prototype would exist when the *load existing* button was activated.

22) The mechanism for prototype decomposition was not described in the previous design.

23) A complete prototype description cannot be entered within the graphic editor.

E. MODIFICATIONS TO THE DESIGN AND THE IMPLEMENTATION

The first ten items have been corrected in the design and the implementation. The eleventh item was partially corrected, the remaining items have been corrected in design only.

1) We changed the frame title of the graphic_editr from dataflow diagram editor in the implementation to **CAPS - GRAPHIC EDI**   in both the design and the implementation.

2) We added a print screen option to the graphic editor. This was implemented by creating a new control option button in the system control group panel of the graphic editor for *print design*. The print design selection causes a screen dump to a file, which is then sent to a printer. This option performed well when the graphic user was displayed on a window device which was physically connected to a Sun server. The file did not survive transfer across the network, when the physical window device was connected to a diskless workstation, due to the size of the transfer file. Since the functionality of the print design button is inoperative on diskless workstations, a further modification was made to determine the physical device that the user is on before displaying the *print design* button. If the user is physically located on a diskless workstation then the button is not provided.

3) We resized the default size of the graphic editor to better utilize the monitor display. The graphic editor now utilizes 100 percent of the screen space.

4) We initialized dynamic memory to NULL in graph.c to correct the segmentation violation. This required systematic debugging to locate the source of the problem. The location of the segmentation violation was in nodes.p. However, we traced it to graph.c where memory was dynamically allocated but not initialized.

5) The tail pixel on inputs, outputs and data-flows was not erased when the rest of the graphic symbol was erased. This occurred when inverting the pixels on the display from the last mouse position to the current mouse position for pixels between the tail and the head of the graphic symbol. This was accomplished with a built in sunview function. To correct this problem the drawing space is redrawn when a symbol is erased. Since the

45

tail pixel is deleted from memory when a symbol is erased, redrawing causes the tail pixel to disappear.

6) A mouse interface panel was created. This panel explicitly denotes the functions of the left, middle and right mouse buttons. The mouse interface panel resides as the top panel of the graphic editor since the designer must know how to interface with the editor before he can make use of the tool. This reduces the size of the drawing space somewhat, but contributes to the usefulness of the graphic editor significantly. In consideration for maintaining maximum area for the drawing space the editing mode panel was reduced in size to eliminate unnecessary wasted space. During the design of the mouse interface panel we also realized that the labels and buttons of the previous design were not consistent. We corrected this as well. A later modification might be to enable the expert user to delete this panel in favor of a larger drawing space.

7) A message panel for editor error messages and warnings was designed into the graphic editor to avoid the possibility of overlaying the design with graphic editor messages. This panel resides immediately above the drawing space.

8) The capability to quit without saving a current session with the graphic editor was implemented. Previously the designer could not quit the graphic editor without storing the design, although two independent buttons are used for these functions. The previous implementation would display a warning message that the graph had not been saved, and would then ignore the request to quit. The designer could quit without storing by using the pull down menu of the graphic editor frame. This design feature essentially encouraged the designer to circumvent the tool. Since it is easily recognized that the

designer might like to quit without storing his current work, this option was integrated into the graphic editor. This was accomplished by using the sunview confirm feature when the quit button is selected and the graph has not been saved. If the designer selects quit without having saved the graph, then a warning message is displayed and a pop-up confirmation sequence occurs. If the designer does want to save the graph before quitting, then he may click the right mouse to cancel the quit request. If he did intend to quit without saving then he may click the left mouse to confirm his request.

9) The operator data components produced by the link statement analyzer were corrected to append the PSDL keyword END to the NewNode product. In the PSDL grammar all SPECIFICATION keywords are bracketed with the keyword END.

10) The previous design and implementation included a command line help facility. Since the graphic editor is an integrated tool within the CAPS environment, and is invoked from within the user interface and not from the command line, this facility was deleted.

11) The delete function was evaluated and partially corrected. The first bug identified was the incorrect use of the C free function for dynamic memory. The free function was applied to memory which was automatically allocated rather than dynamically allocated. Kernighan and Ritchie state that "it is a ghastly error to free something not obtained by calling calloc or malloc" [26]. This was corrected by eliminating the free function calls within the file graph.c. The second bug related to the inconsistent behavior of the delete function. It appeared as if all objects which are represented by arrows in the graphic editor such as inputs, outputs, data-flows and self loops should be able to be deleted by

pressing the right mouse button on the tail or the head of the object. But when the right mouse was pressed on the head of a self loop the associated operator and all of its inputs, outputs, data flows and self loops were deleted. Investigation revealed that when the right mouse was pressed, if the mouse was located on an input, output, data-flow or self loop the respective line would be deleted. If the right mouse was pressed and the graphic editor did not determine that the mouse was positioned on an arrow, then it checked to see if the right mouse was within an operator. When the right mouse was pressed on the head of a self loop, and if the head of the self loop was located within an operator, the graphic editor would never recognize that the mouse was positioned on a arrow so the delete operations for an operator would be activated. The implementation for drawing arrows in the drawing space uses variable names which correspond to points within a Cartesian coordinate system. The algorithm which creates the data structure which represents an arrow always uses $(x1,y1)$ as the tail and $(x2,y2)$ as the head. The algorithm for checking if the right mouse is on a arrow, only checks if the mouse is positioned on the head or the tail of an arrow. When a self loop is created $(x1,y1)$ is the tail, but $(x2,y2)$ is not actually the head. The actual Cartesian coordinates of the points for a self loop are shown in Figure 5-5.
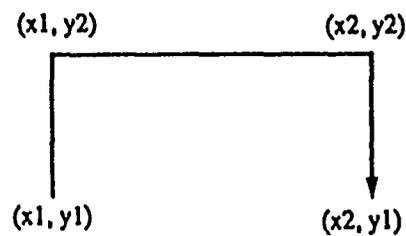
Figure 5-5. Self Loop Locations

48

The head of a self loop is actually the point (x2,y1). The values stored as the head of a self loop in the current implementation are (x2,y2). Currently a self loop may be deleted by selecting (x1,y1) or (x2,y2) with the middle mouse button. This causes the inconsistencies noted during testing. Since the algorithm does not recognize (x2,y1) as the head of an arrow, if the arrow intersects an operator, then the operator and all related objects are deleted (including the self loop). We decided that limiting the recognition of an arrow to the end points was a poor design decision. Rather than correct the implementation to store (x2,y1) as the head of the self loop, we chose to temporarily resolve this problem by redefining the mouse interface panel to state explicitly that deletion of self loops is performed only when the mouse is positioned on the tail of a self loop. The correct way to fix this problem is universal to all objects represented by arrows. The graphic editor should be able to recognize all points along a line. When any object in the drawing space is selected for deletion, the object should be highlighted in some manner. A verification mechanism should also be utilized to avoid inadvertent deletions. A pop-up window which requires that the designer verify the deletion should used whenever a delete operation is requested.

12) The input, output and data-flow symbols need to be clipped. The type of symbol is specified, before the location or end points of the symbols are specified. The syntactic correctness is checked using the symbol's related operators. The location of the operators is known to the graphic editor. The graphic editor should adjust the head of the input and data-flow symbols to the borders of their respective operators. The tails of the output and data-flow symbols should be adjusted to the borders of their respective operators.

49

This will greatly reduce the precision required of the designer to create a graphic representation of the prototype with a neat appearance.

13) The tail then head constraint on the selection of input and output mouse selections should be removed. Since a *draw input* or *draw output* has been selected before the user specifies the end points, the graphic editor should be able to determine the appropriate head and tail position, and should permit the designer to select the end points in either order.

14) The additional capability to resize objects should be implemented. Currently the designer must delete then redraw an object which he would like to resize. The current deletion of operators removes all related textual information and adjacent arrows, which further compounds this problem. Without a resizing capability, the designers might eventually learn to always draw their designs on a small scale to ensure that they do not run out of drawing space. This type of compensation would encourage poor utilization of the drawing space and the tool. A resizing capability will enhance the friendliness and usefulness of the graphic editor significantly. The resizing of objects may be implemented by allowing the designer to select the border of an object by pressing down on the left mouse button, dragging the mouse while the object rubber bands, and when the object obtains the desired size, releasing the left mouse button to effect the change in size of the object.

15) The implementation of the additional capability to relocate objects in the drawing space is required. The lack of the ability to relocate objects will have the same negative impact on a designer as the inability to resize objects. The designer may learn to over

50

compensate thus reducing the effectiveness of the graphic editor. The relocation of objects may be implemented with the following process. The designer should select the border of an object to relocate by pressing down on the left mouse button. Then he should drag the mouse to move the object. When the object has the new desired location the designer may release the left mouse button to effect the change.

16) The ability to invert data-flows would be a nice enhancement. This should be implemented by copying the existing end points, rewriting them in reverse order into the permanent data structure, and then redrawing the display.

17) The correct translation of keyboard back space input is essential since the output of the graphic editor is used as input for other tools in the environment. The file nodes.p must be modified to check for the back space control character, and then adjust the characters in the text buffer appropriately.

18) The implementation of name analysis is an essential feature. A data structure should be created which contains the names assigned to data flows and operators. Before new names are accepted by the graphic editor, the names should be compared with existing names. The graphic editor should analyze the object being named and ensure that conflicts or multiple declarations are not defined. If the new name violates these conditions, then a message should be displayed in the message panel and the designer should redefine the name.

19) The modification of names and time constraints independently of their associated operators requires that the graphic editor recognize the boundary of the names and time constraints associated with an operator when selected with the mouse. Functions should

be provided in the file graph.c which control the modification of these objects whenever they are selected.

20) Currently a time constraint is required by the graphic editor for all operators. This requirement should be removed. If the designer does not enter a time constraint then a default value of zero should be assigned. Currently, units are required for all time constraints. Since the PSDL language uses milliseconds as a default value, the function within the graphic editor which enforces the entry of units, should be modified to allow the designer to omit this information. If the designer does not explicitly enter the units of time then the default value should be assigned.

21) The previous design decision to have the user interface retrieve the necessary reconstruction information from the design database and store the information into a file called graph.pic, and to then have the designer select *load existing* after the graphic editor becomes available, separates the retrieval function into two different interfaces. This function should be entirely controlled by either the user interface or the graphic editor. Our design modification is to place this function entirely within the graphic editor. This should occur with the following sequence of operations.

a. The designer selects the *load existing* button.

b. A pop-up listing is displayed in the drawing area which contains the names of the existing designs. These designs should be ordered such that the most recently developed designs occur in the listing before the other designs. This ordering assumes that designs to be modified are more likely to be designs which were incompletely reconstructed or pertain to the most recent development project.

c. The designer should select a design by typing the name with the keyboard into a small window space or by pointing at a name with the mouse and pressing a mouse button to make the selection.

d. The graphic editor makes a copy of the design file with a suffix reserved for backup fies.

e. The design is loaded into the graphic editor.

f. If the designer selects *store* with this design, then he is queried as to his desires for the backup copy. He should be able to rename or remove the old copy. Eventually the design database should perform version control and manage all of the versions of a prototype.

g. If the designer selects *quit* without having stored the current version of the design, then the edited version is removed and the name of the backup copy is restored to its original name.

Consistency of design indicates that the graphic editor should control the naming of all graphic designs. Naming should occur when the *store* button is selected. Until the design database is implemented, the directory /caps/prototypes/ should be used to contain the prototype designs.

22) The decomposition process of a PSDL prototype design was not previously described. The existing data structures currently used in the graphic editor should be modified to add another dimension. This dimension would generate a linked list of operators to their decompositions. Three new buttons search, decompose and compose should be added to the system control group panel. The search button interacts with the

Database Management System. The implementation of this function should provide the user with the capability to retrieve both prototype designs and reusable software components. This button has been added to the graphic editor layout but its functionality has not yet been implemented. The details of these operations are further defined in a later chapter. When the decompose button is activated, the user selects an operator in the current design, a *link* from the operator in the current data structure is created, and the drawing space is cleared. The user may now construct the decomposition of the selected operator. Information related to an operator must be consistent with the decomposition of that operator. For example, if an operator has a defined input, then when the operator is decomposed that same input must appear as an EXTERNAL input in the decomposition of the operator. If the designer has defined adjacent inputs, outputs, data streams, time constraints or any other related information prior to selecting an operator for decomposition, this information should be visible to the designer when constructing the decomposition. The graphic editor must ensure that the information related to an operator is consistent in both the decomposition and composition of the operator. When the compose button is activated the drawing space is cleared and the design which contains the parent operator is displayed. The decompose and the compose buttons have been added to the graphic editor, but their functionalities have not yet been implemented.

23) We believe that a user should be able to describe a PSDL design completely from within the graphic editor. Better utilization of sunview capabilities within the graphic editor and enhancements to the current data structures used within the graphic editor support this modification. The use of pop-up menus and the use of multiple windows can provide an interface which avoids clutter and information overload. The current set of

graphic symbols remains sufficient. Optional PSDL constructs which relate to an operator and are not currently supported by the graphic editor can be displayed within a pop-up menu. Selections made within a pop-up menu can generate dialogue boxes where the user describes the optional constructs. The dialogue boxes should be customized to support effectively each optional construct. A few examples of these constructs include control constraints, informal descriptions, and formal descriptions. Mechanisms used within these dialogue boxes can be question and answer, mini syntax directed editors, or simple text editors. This information represents an annotation view [27] of the operator. It will not be visually persistent but will be attached to the operator. The existing data structures can be expanded to manage these constructs.

The layout of the graphic editor as it is currently implemented, with a simple prototype design, is shown in Figure 5-6. The functional components which comprise the graphic editor are contained in Appendices C through F.

Figure 5-6. Current Graphic Editor

VI. THE SYNTAX DIRECTED EDITOR

Language-based editors or syntax directed editors are editors which are tailored to a specific language. These editors use the grammar, structure and static semantics of a language to assist a user in writing correct programs. These editors generally enforce syntactically correct programs by providing program segment templates which contain legal alternatives for the specific language and prohibit illegal constructs. They may also combine plain text editing with incremental parsing techniques to ensure that only syntactically correct program fragments are entered.

A. LANGUAGE-BASED EDITOR GENERATORS

Porter [11] performed a comparison of two predominant editor generators currently in use for developing language based editors. These tools were the Cornell Synthesizer Generator and the GANDALF ALOE Generator. Porter stated that the Cornell Synthesizer Generator [28,29] was the more appropriate tool for our purposes. The Gandalf tool provides an environment which permits team development of system software. It exceeds the scope of the Cornell Synthesizer Generator by providing both programming and system development environments. The desired PSDL editor is designed as a tool within the CAPS environment. CAPS provides its own system development capabilities. The development of a PSDL editor using the GANDALF tool would underutilize the GANDALF tool.

Porter established the feasibility and recommended the use of the Cornell Synthesizer Generator for the development of a PSDL editor for CAPS. We decided to follow Porter's recommendation and did not conduct any further evaluation of editor generators.

B. THE CORNELL SYNTHESIZER GENERATOR

The Synthesizer Generator creates a language-specific editor from an editor specification which describes the language. The editor specification defines the abstract syntax, context-sensitive relationships, display format, concrete input syntax, and transformation rules for the editor. These specifications are written by the editor-designer using the Synthesizer Specification Language (SSL).

The Cornell Synthesizer Generator is written in C and runs under UNIX. The central components of an editor created from the Cornell Synthesizer Generator are the editing kernel and the generator proper.

The editing kernel consists of four subcomponents which are common to all generated editors. These subcomponents are an attributed-tree module, the SSL-expression interpreter, the editor module, and the display module. The attributed-tree module contains a set of operations for manipulating attributed trees. An example is the incremental algorithm for updating a tree's attribute values after it has been modified. The SSL-expression interpreter is invoked by the attributed-tree module. This occurs when a new value of an attribute instance is to be computed and when a transformation has been applied to an attributed tree. The editor module provides the capabilities for

manipulating objects within an editor. These objects are contained in a collection of buffers. The editor module provides the system commands such as those for structural editing and textual editing. The display module provides the support for video-display terminals, bit-mapped workstations and mice.

The generator proper generates editors from editor specifications. It consists of a shell program and the SSL translator. The shell program, *sgen*, coordinates the activities of the SSL translator with the UNIX utilities, such as lex, yacc and cc, which are employed during the process of creating an editor. The SSL translator processes the SSL source which represents the editor specifications. Four unique subcomponents are created for any generated editor. The subcomponents are the editor's grammar tables, scanner, parser, and sequences of byte-codes which are the internal representation of SSL expressions.

The SSL reserved words which may not be used by an editor designer for any other purpose are shown in Figure 6-1.

and	as	default	demand	end
exported	ext_computers	false	forall	foreign
in	inh	inherited	left	list
let	local	nil	nil_attr	nonassoc
on	optional	parse	prec	readonly
repeated	right	root	sparse	store
style	syn	synthesized	transform	true
typedef	unparse	view	with	

Figure 6-1. SSL Reserved Words

C. PREVIOUS DESIGN AND IMPLEMENTATION

Porter generated some partial specifications for a PSDL editor, but the specifications were completely untested. After reviewing his work, we were unable to perform any editing with those specifications. We chose to define our own approach to development and to disregard the previous implementation.

D. DEVELOPMENT OF THE PSDL EDITOR

The modular development of an editor using the Cornell Synthesizer Generator consists of editor specifications which may be divided into six major components:

- abstract syntax declarations
- unparsing declarations
- lexical declarations
- concrete input syntax declarations
- attribute declarations and equations
- templates and transformations

Modular construction permits the abstraction of the requirements of each component, enhances comprehensibility and incorporates effective extensibility for an editor. Good engineering practice was also employed by first defining a small subset of the PSDL language for the initial editor, while familiarizing ourselves with the specifics of the process for generating an editor. The purpose and implementation details for the construction of each module is described using the initial subset of the PSDL grammar shown in Figure 6-2. Some of the PSDL nonterminals have been eliminated or are treated as terminals here in our subset. The complete PSDL grammar is contained in Appendix A.

```
psdl
    - (component)
component
    - data_type | operator
data_type
    - "type" id type_spec
operator
    - "operator" id operator_spec
type_spec
    - "specification" (type_decl) "end"
operator_spec
    - "specification" (interface) "end"
interface
    - attribute (reqmts_trace)
attribute
    - input | output
input
    - "input" type_decl
output
    - "output" type_decl
reqmts_trace
    - "by requirements" id_list
type_decl
    - id_list ":" id
id_list
    - id ("," id)
id
    - letter (alphanumeric)
alphanumeric
    - letter | digit
letter
    - "a..z" | "A..Z" | "_"
digit
    - "0..9"
```

Figure 6-2. Original Subset Of PSDL Grammar

The descriptions of the SSL modules embodies lessons learned during the generation

of the editor that may be applied to the generation of any editor. We decided to make the

initial editor for use on a video terminal rather than a workstation due to the availability

of local resources. The final version of the editor within CAPS is expected to utilize graphics with a mouse on a Sun Workstation.

Before writing any SSL specifications we recommend that the editor designer define an intermediate grammar. There may exist several conditions in the BNF dialect of the grammar which, if implemented literally, will probably produce an editor which is cumbersome to the user. The intermediate grammar will not change the meaning of the original grammar but will provide a more natural transition between the BNF grammar and the SSL declarations.

Nonterminals that carry no specific semantic meaning should be eliminated. These nonterminals may exist in the BNF dialect of the grammar to enhance readability and understandability. When used in the abstract syntax they produce unnecessary editing steps and depth in the derivation trees. An example of this in our initial PSDL grammar is the production for *component* which resolves to a *data_type* or an *operator*. We redefined *component* in our intermediate grammar to be a *"type" id type_spec* or a *"operator" id operator_spec*.

BNF dialects and SSL differ in the way that optional occurrences of nonterminals are treated. A BNF dialect uses a mechanism to state that an instance of a nonterminal is optional. SSL uses a property declaration for the nonterminal to state that it is optional. Thus all occurrences of that nonterminal are optional. We recommend that the editor designer identify nonterminals in the BNF dialect of the grammar which have both required and optional occurrences. A convention for creating new nonterminals in the intermediate grammar to differentiate between optional and required instances of the

nonterminals should be established. The new optional nonterminals carry the same semantic meanings as the original nonterminals. We created new nonterminals for optional instances and used a convention of prefixing *optional_* to the name of the new nonterminal for optional occurrences. An example of this issue occurs in our original PSDL subset with the *type_decl* production. We created a new nonterminal *optional_type_decl* in our intermediate grammar. We also decided to maintain an expressive regularity by renaming all nonterminals which only had optional occurrences in the original grammar with the same *optional_* prefix.

BNF dialects and SSL also differ in the way that optional instances of sequences of terminals and nonterminals occur. In SSL a property declaration is used to specify that all occurrences of a nonterminal are lists, or optional lists. Lists in SSL are treated as binary trees. An SSL list must have exactly two operators. One operator is a nullary operator constructing an empty list and the other is a right recursive binary operator adding a new list element to a given list. A list in SSL must contain at least one element. A production *list = list_item {list_item}* would be reflected as a list in SSL. Optional lists permit an instance of the list which is empty. A production such as *list = {list_item}* would be reflected as an optional list in SSL. We recommen¹ that the editor designer identify whether sets in the original grammar are lists or optional lists. We renamed the optional list nonterminals with the prefix *optional_list_*. In our original grammar *interface* is an optional list and was renamed to *optional_list_interface* in our intermediate grammar. In our original grammar *id_list* is an list. Our intermediate grammar for the PSDL subset is shown in Figure 6-3.

63

```
psdl
    ~ list_component
list_component
    ~ "type" id type_spec | "operator" id operator_spec
type_spec
    ~ "specification" optional_type_decl "end"
operator_spec
    ~ "specification" optional_list_interface "end"
optional_list_interface
    ~ attribute optional_reqmts_trace
attribute
    ~ input | output
input
    ~ "input" type_decl
output
    ~ "output" type_decl
optional_reqmts_trace
    ~ "by requirements" id_list
type_decl
    ~ id_list ":" id
optional_type_decl
    ~ id_list ":" id
id_list
    ~ id ("," id)
id
    ~ letter (alphanumeric)
alphanumeric
    ~ letter | digit
letter
    ~ "a..z" | "A..Z" | "_"
digit
    ~ "0..9"
```

Figure 6-3. Intermediate PSDL Subset

1. Abstract Syntax Declarations

The abstract syntax is the core of the editor specification. It is defined as a set of grammar rules. An object in the resultant editor is represented by a derivation tree which is constructed based on the grammar.

The abstract syntax declarations may easily be defined from the intermediate grammar. The literals of the grammar are not considered at this stage as they do not represent parts of a derivation tree. The abstract syntax for the PSDL subset is shown in Figure 6-4.

Nonterminals with special properties such as optional, list or optional list are denoted by using the appropriate property declarations in the abstract syntax. Optional nonterminals must contain a nullary operator in addition to any other desired operators.

The abstract syntax declarations are a collection of productions. In SSL productions have the form $x0 : op(x1\ x2\ ...\ xk)$, where op is an operator name and each xi is a nonterminal of the grammar. The nonterminal is also referred to as a *phylum*. The phylum associated with a given nonterminal is the set of derivation trees that can be derived from it by using operators. These derivation trees are referred to as *terms*. The operators identify the production instances in a derivation tree.

The SSL grammar rule acts like a context-free production $x0 \rightarrow x1\ x2\ ...\ xk$. All operator names must be unique. The operator of a production distinguishes it from the other alternatives provided by the left hand side phylum. One phylum in the abstract syntax declarations must be distinguished as the root phylum. All editable objects in the editor are terms of the root phylum.

Phyla contain a *completing term* and a *placeholder term*. The same term may be both a *completing term* and a *placeholder term*. The first operator declared for each phylum is the *completing operator*. The *completing operators* construct default representations for the phylum called the *completing term*. An instance of the appropriate

65

```
root psdl_components;
list psdl_components;
psdl_components
    :  PsdlNil()
    |  PsdlPair(component psdl_components);
component
    :  NoComponent()
    |  Data(id type_spec)
    |  Op(id operator_spec);
operator_spec
    :  OpSpec(optional_interface);
type_spec
    :  TypeSpec(optional_type_declaration);
optional list optional_interface;
optional_interface
    :  InterFaceNil()
    |  InterFaceList(attribute optional_interface);
attribute
    :  EmptyAttr()
    |  Input(input optional_requirements)
    |  Output(output optional_requirements);
optional optional_requirements;
optional_requirements
    :  ReqmtsTraceNone()
    |  ReqmtsPrompt()
    |  ReqmtsTrace(id_list);
input
    :  InputTypeDecl(type_decl);
output
    :  OutputTypeDecl(type_decl);
type_decl
    :  TypeDecl(id_list type_name);
optional optional_type_declaration;
optional_type_declaration
    :  OptTypeDeclNil()
    |  OptTypeDecl(id_list type_name);
type_name
    :  TypeName(id);
list id_list;
id_list
    :  IdNil()
    |  IdPair(id id_list);
id
    :  IdNull()
    |  Id(IDENTIFIER);
```

Figure 6-4. Abstract Syntax

completing term resides at each unexpanded occurrence of a phylum in a derivation tree. The *completing term* of a list phylum differs in that the *completing term* is the singleton list constructed by applying the binary operator to the completing term of its left argument phylum and to the list's nullary operator. *Placeholder terms* identify locations where subterms may be inserted. The relationship of completing terms and placeholder terms varies dependent upon the property declarations for a phylum. For ordinary phyla and list phyla the same term is both the completing term and the placeholder term. For optional phyla the completing term is constructed from its first nullary operator. The placeholder term is constructed from the first operator which is not used to construct the completing term. For optional list phyla the completing term is constructed from the nullary operator, and the placeholder term is the singleton list constructed by applying the list's binary operator to the completing term of the list's left child and to the list's nullary term.

2. Unparsing Rules

The next step in constructing an editor is to define the display representation. The display representation is described by a collection of unparsing rules. The unparsing rules define the behavior of the editor with respect to the abstract syntax. This module contains specifications for the display format and for denoting which nodes in the abstract syntax tree are selectable and which productions of an object are editable.

The SSL form for unparsing rules is *phylum : operator [unparsing syntax]* where *phylum* and *operator* correspond on a one to one basis with the abstract syntax. The unparsing syntax includes a selection symbol which corresponds to the left hand side

67

phylum and a selection symbol for each node on the right hand side in the order in which the nodes occur in the abstract syntax.

There may be locations in the derivation tree which need not be visible to the user of our editor. The selection symbol which denotes a node as a *resting place* in the derivation tree is a @. We denote nodes that we do not wish to be selectable, which means they are not resting places, by the ^ selection symbol. The selection symbol for the left-hand-side-phylum is separated from the right-hand-side-nodes by a : if we do not wish an object to be editable, or by a ::= if we do intend for the object to be editable. A node in the tree is an instance of two phylum occurrences. It occurs in the right-hand-side and in the left-hand-side. If either occurrence is represented with a @ symbol then the node represented the phylum will be designated as selectable otherwise it is not selectable. This characteristic necessitates the development of another convention. The editor designer should choose a convention for the insertion of resting places, either in place for the left-hand-side phylum or the right-hand-side phylum. We chose the left-hand-side phylum as our convention since the resting places were more easily recognized and fewer @ symbols were required in the unparsing rules. The trade-off with this choice is that if the editor-designer desires that a phylum be a selectable in one subtree and not selectable in another subtree then the unparsing rules must be modified to describe the right-hand-side occurrence as a resting place and the left-hand-side as not a resting place. This occurs infrequently in our PSDL language.

The syntactic sugar of the language is interspersed within the unparsing syntax in the form of tokens. These tokens are enclosed within double quotes. Display

68

formatting such as newlines, tabs and back tabs may also be included within double quotes. The SSL display formatting commands are shown in Figure 6-5.

Formatting Command	Meaning
%t	move the left margin one indentation unit to the right
%b	move the left margin one indentation unit to the left
%n	newline, return to the current left margin
%l	return to the current left margin and overprint
%1	move to column one of the same line and overprint
%T	move right to the next tab stop
%M(c)	move right to column c, where c is a positive integer
%o	optional newline, return to the current left margin
%c	same as %o, but either all or no %c in a group are taken
%(	beginning of an unparsing group
%)	end of an unparsing group
%[	same as %t%(
%]	same as %)%b
%S(style-name	enter the named style
%S)	revert to the previous style
%%	display a %

Figure 6-5. SSL Display Formatting Commands

Another convention should be established with regard to the placement of tabs and newlines. The consideration is whether to place the newlines in the front of the unparsing rules or at the end of the unparsing rules, and the level at which to place the tabs and the back tabs. The effect of the formatting commands for tabs and back tabs are realized upon the next occurrence of a newline. We chose to place the newlines at the front of the unparsing rules, and tabs and back tabs in the parent rules. The unparsing rules for our PSDL subset are shown in Figure 6-6.

```
psdl_components
    :  PsdlNil            (@:)
    |  PsdlPair           (@:^("%n")^) ;
component
    :  NoComponent        (^:"%n(component)"^)
    |  Op                 (^:"%nOPERATOR ^"^)
    |  Data               (^:"%nTYPE ^"^) ;
id
    :  IdNull             (@::="<identifier>")
    |  Id                 (@::="^) ;
operator_spec
    :  OpSpec             (^:"%nSPECIFICATION%t"^"^%b%nEND") ;
type_spec
    :  TypeSpec           (^:"%nSPECIFICATION%t"^"^%b%nEND") ;
optional_interface
    :  InterFaceNil       (@:)
    |  InterFaceList      (@:^()@) ;
optional_requirements
    :  ReqmtsTraceNone    (@:)
    |  ReqmtsPrompt       (@:"%n(requirements)")
    |  ReqmtsTrace        (@:"%nBY REQUIREMENTS%t%n"^"^%b") ;
attribute
    :  EmptyAttr          (^:"%n(interface)")
    |  Input              (^:"%nINPUT"^"^%t"^"^%b")
    |  Output             (^:"%nOUTPUT"^"^%t"^"^%b") ;
input
    :  InputTypeDecl      (^:"%t%n"^"^%b") ;
output
    :  OutputTypeDecl     (^:"%t%n"^"^%b") ;
type_decl
    :  TypeDecl           (^:"^" : "^") ;
optional_type_declaration
    :  OptTypeDeclNil     (@:"%n(optional type declaration)")
    |  OptTypeDecl        (@:"%n"^"^ : "^") ;
type_name
    :  TypeName           (^:^) ;
id_list
    :  IdNil              (@::=)
    |  IdPair             (@::=^(", ")@) ;
```

Figure 6-6. Unparsing Rules

3. Lexeme Declarations

The next step in the development of an editor is to define the lexical rules. The form of a lexeme declaration is *phylum-name : lexeme-name < regular-expression >;* This declaration states that all strings generated by the given regular expression are in phylum-name. The regular expression is separated from the closing angle bracket by at least one blank character. The regular expression may contain embedded blank characters by explicitly escaping the blank with a back slash. The lexeme-name will be used in the concrete input definitions. The regular expressions permitted in the SSL lexeme declarations are generally the same regular expressions which are accepted by the UNIX lexical analyzer generator lex. Figure 6-7 contains guidance for acceptable regular expressions.

Expression	Meaning
c	the character c
"c1c2c3"	the string c1c2c3
[c1c2c3]	the character c1,c2 or c3
[c1-c2]	any of the characters from c1 through c2
[^c1c2c3]	any character but c1,c2 and c3
^e	an e at the beginning of a line
e$	an e at the end of a line
e?	an optional e
e*	0 or more instances of e
e+	1 or more instances of e
e1e2	an e1 followed by an e2
e1|e2	an e1 or an e2
(e)	an e
e1/e2	an e1 but only if followed by an e2
e{n1,n2}	n1 through n2 occurrences of e

Figure 6-7. Regular Expressions

Lexeme declarations form an ordered list. During lexical analysis in an editor, this order influences the recognition process. When more than one regular expression matches a string, the longest match is selected. If several rules match with the same number of characters, then the first declaration which matched in the specification is selected. The lexeme declarations for our PSDL subset are shown in Figure 6-8.

```
IDENTIFIER: IdentLex< [a-zA-Z][a-zA-Z_0-9]^ >;
```

Figure 6-8. Lexeme Declarations

---

4. Attribute Declarations

The next step in the development of an editor is to define the attribute declarations. The attribute declarations define the association between the abstract syntax and the concrete input syntax. SSL attribute declarations associate an attribute with the name of a phylum. They also describe the type and the source of the attribute as either synthesized or inherited. The form of an SSL attribute declaration is *phylum (source attribute type;)*; This file also contains entry declarations which establish the correspondence between the selections in the abstract-syntax tree which are to be edited and the entry points within the input syntax. The form of an SSL entry declaration is *p ¬P.t;*. This example indicates that when a selected component of the program is a member of phylum *p*, input is to be parsed to determine if it is a member of *P*. If it is, then attribute *t* should be inserted in the abstract-syntax tree, and should replace the current selection. The attribute declarations and their entry declarations for our PSDL subset are shown in Figure 6-9.

72

```
Ident          (synthesized id t;);
Id_list        (synthesized id_list t;);


id             ~ Ident.t;
id_list        ~ Id_list.t;
```

Figure 6-9. Attribute Declarations

5. Concrete Input Syntax

The concrete input syntax of a language to be used for text editing is defined in terms of a concrete input grammar. It comprises the rules which specify the set of syntactically well-formed strings, the structure of parse trees, and attribute equations that specify the translation of the parse trees into terms of the abstract syntax.

The attribute equations specify how values of synthesized attributes of the left-hand side nonterminals are computed in terms of their inherited attributes and synthesized attributes associated with nonterminals on the right-hand side. They also determine how inherited attributes of right-hand side nonterminals are constructed from inherited attributes of the left-hand side nonterminals and from synthesized attributes of left siblings.

The minimum syntax required for an editor must provide for all language constructs which are entered as text via the keyboard. Ultimately, the concrete input syntax should recognize the entire language to permit the user to read existing objects into a text file and then edit those objects. The concrete input syntax for our PSDL subset is shown in Figure 6-10.

73

```
Ident       ::= (IDENTIFIER)
                (Ident.t = Id(IDENTIFIER););


Id_list     ::= (Ident)
                (Id_list.t = (Ident.t::IdNil);)
              | (Ident ',' Id_list)
                (Id_list$1.t = (Ident.t::Id_list$2.t););
```

Figure 6-10. Concrete Input Syntax

The left-hand side phyla name are separated from the right-hand side symbols with a ::= symbol. Single characters may be enclosed in single quotes on the right-hand side of the parsing declarations. Different occurrences of the same nonterminal are distinguished by appending a number, such as $1 and $2, which reflects their occurrence in the rule. The second rule in our syntax specifies that a list of identifiers is separated by a comma in the concrete syntax and that the lexeme values parsed are concatenated to a list (using the predefined concatenation operator ::) whose value is stored in the synthesized attribute t of the nonterminal Id_list.

6. Templates and Transformations

The next step is to define the templates and transformations. Transformations may define templates which are inserted whenever the selection is a placeholder or may compute a replacement value dependent upon the former value of the selection. We only defined template transformations for our first editor. These declarations specify the restructuring of objects when a current selection matches a given pattern. The general form

74

of a transformation declaration is *transform x0 on transformation-name <x0> :
operator(<x1>,...,<xn>);* , where *transform* and *on* are SSL reserved words, *operator* is
an operator and *x0* through *xn* are productions in the abstract syntax. The template
transformations for our PSDL subset are shown in Figure 6-11.

```
transform component
    on "type"
        <component> :
        Data(<id>,<type_spec>),
    on "operator"
        <component> :
        Op(<id>,<operator_spec>);

transform attribute
    on "input"
        <attribute> :
        Input(<input>,<optional_requirements>),
    on "output"
        <attribute> :
        Output(<output>,<optional_requirements>);

transform optional_requirements
    on "enter_requirements"
        <optional_requirements> :
        ReqmtsTrace(<id_list>);

transform optional_type_declaration
    on "enter_declaration"
        <optional_type_declaration> :
        OptTypeDecl(<id_list>,<type_name>);
```

Figure 6-11. Template Transformations

E. DESIGN ISSUES OF THE COMPLETE PSDL EDITOR

The PSDL Syntax Directed Editor described here and used in CAPS is a very
simplistic editor with regard to the level of sophistication capable in an editor generated

75

from the Cornell Synthesizer. The simplistic design of our PSDL editor was intentional. We have purposely turned off features which may normally be included in an editor generated from the Cornell Synthesizer Generator. We have purposely documented only a very small subset of the available editor commands. The subset of editor commands provided in our on-line documentation is sufficient to use the PSDL editor as we intended. In an environment which contains multiple tools which interact with the user, we believe that the interfaces must be kept simple and should be as consistent amongst the different tools as possible. We felt that the trade-off in the complexity of the editor's user interface with the additional capabilities provided by the Cornell Synthesizer Generator was not warranted for our application.

Our editor is consistent with the principle that a user interface should always provide feedback for any user input. The display always changes in appearance any time the user enters an editor command or any other keyboard input.

We have updated the PSDL grammar during the design and implementation of the PSDL editor. There were four changes in the grammar which were not simple factorizations but that actually changed the *meaning* of the language. The first change simply made the grammar more regular. PSDL primary objects such as operators and data types have two parts: a specification part and an implementation part. The specification construct begins with **SPECIFICATION** and ends with **END**. The implementation part may consist of a *psdl implementation* or an *ada implementation*. The *psdl implementation* construct begins with **IMPLEMENTATION** and ends with **END**. The *ada implementation* construct began with **IMPLEMENTATION ADA** and contained an Ada program. An *ada implementation* did not require an **END** to bracket the construct. We

76

decided to change the grammar to require that an *ada implementation* end with an **END** to increase the regularity and syntactic consistency of the language.

The second, third and fourth changes to the grammar involved a similar issue. The rules for data_flow_diagram, control_constraints and constraint permitted that the smallest correct use of the rules resulted in unsatisfactory conditions. The last three changes to the rules resulted in required subsets of the original grammar. The rules which relate to data_flow_diagram and control_constraints are shown in Figure 6-12.

```
psdl implementation
      = "implementation" data_flow_diagram [streams] [timers]
      [cor' rol_constraints] [informal_desc] "end"
data_flow_diagram
      = "graph" (link)
control_constraints
      = "control constraints" (constraint)
```

Figure 6-12. Original PSDL Rules

The smallest correct psdl implementation which resulted from these rules is shown in Figure 6-13.

```
IMPLEMENTATION
    GRAPH
END
```

Figure 6-13. Smallest PSDL Implementation According To Previous Grammar

This yields an empty implementation. We decided that an empty implementation should not be permitted. The grammar was changed to require that a psdl implementation contain at least one link statement. The change in the grammar is shown with its corresponding smallest construct is shown in Figure 6-14.

```
data_flow_diagram
   = "graph" link (link)
```

```
IMPLEMENTATION
   GRAPH
   id "." id [":" time] "->" id
END
```

Figure 6-14. Updated Psdl Implementation Rule And Its Smallest Construct

The control constraints construct is optional in the psdl implementation rule. Previously when selected, it could consist of keywords only, as shown in Figure 6-15.

```
IMPLEMENTATION
   GRAPH
      id "." id [":" time] "->" id
   CONTROL CONSTRAINTS
END
```

Figure 6-15. Smallest Control Constraint According To Previous Grammar

We changed the PSDL grammar so that if a control constraint construct was selected, then the construct must contain at least one constraint. The current control constraint rule and its smallest corresponding result are shown in Figure 6-16.

```
IMPLEMENTATION
    GRAPH
        id "." id [":" time] "->" id
    CONTROL CONSTRAINTS
        OPERATOR id
END
```

Figure 6-16. Current Smallest Control Constraint In Context

The fourth change in the grammar concerned the *triggered* option in the constraint rule. The problem we found with this rule was similar to the problem with the previous smallest constraint rule. The previous constraint rule and an example of the smallest construct using the triggered option is shown in Figure 6-17.

```
constraint
    = "operator" id
    ["triggered" [trigger] ["if" predicate] [reqmts_trace]]
    ["period" time [reqmts_trace]]
    ["finish within" time [reqmts_trace]]
    (constraint_options) .



CONTROL CONSTRAINTS
    OPERATOR id
    TRIGGERED
```

Figure 6-17. Previous Triggered Option

79

We changed the triggered option so that if it was selected it requires either a trigger or an optional trigger followed by a required "if" predicate. The current constraint rule with the smallest possible triggered options are shown in Figure 6-18.

```
constraint
   = "operator" id
   ("triggered" (trigger | [trigger] "if" predicate) [reqmts_trace])
   ("period" time [reqmts_trace])
   ("finish within" time [reqmts_trace])
   (constraint_options)

CONTROL CONSTRAINTS
   OPERATOR id
   TRIGGERED
   BY ALL a,b

CONTROL CONSTRAINTS
   OPERATOR id
   TRIGGERED
   IF a = 0
```

Figure 6-18. Current Triggered Option

The complete PSDL Grammar shown in Appendix A is the updated grammar which reflects the changes we have described. The implementation of our PSDL editor is contained in Appendices G through L.

F. INTEGRATION

The files which comprise the functional components of the PSDL editor and related documentation files reside mostly in the directory /n/suns2/work/caps/syntax_editor. These components are:

pev	executable terminal psdl editor
pew	executable sunview psdl editor
psdl.as.ssl	abstract syntax for psdl editor
psdl.ad.ssl	attribute definitions for psdl editor
psdl.ci.ssl	concrete input for psdl editor
psdl.lex.ssl	lexical file for psdl editor
psdl.tt.ssl	template transformations for psdl editor
psdl.up.ssl	unparsing rules for psdl editor
Makefile	shell script to generate an executable PSDL editor from the psdl.*.ssl specifications.

An additional file which initializes the syntax directed editor is:

/n/suns2/work/caps/.syn_profile

The on-line manual page instructions for pev is located at:

/n/suns2/usr/man/manl/pe.l

The Cornell Synthesizer Generator is located at:

/n/suns2/usr/suns2/local/syn

When the syntax directed editor is required the user interface creates a new process which executes the PSDL editor concurrently with the user interface and any other potentially active processes. When the designer saves the PSDL prototype from within the editor it is initially saved in the caps directory. The user interface directory currently simulates the design database and the software database.

G. USING THE PSDL EDITOR

When a user selects the syntax directed editor from within the CAPS construction menu, he is given the option to view the on-line user's manual. If instructions are requested, then the following description is provided using the UNIX *man* facility. These instructions may also be accessed outside of CAPS by executing *man pe*.

81

NAME

pe - PSDL syntax directed editor

SYNOPSIS

pe

DESCRIPTION

pe is a syntax directed editor for the prototype system description language (PSDL). This editor was designed to be used within the Computer Aided Prototyping System (CAPS). The editor provides a simple, mostly regular user interface which guides the user painlessly through the structure of a correct PSDL program.

There are 3 primary modes of interaction with the editor. Each mode is visually distinctive.

1. Required Keyboard Input is denoted by a token contained in angle brackets, such as <identifier>.

2. Required Construct Choices are contained within square brackets, such as [implementation]. When a construct choice is required, the user selects an alternative by pressing the tab key followed by typing the alternative or typing an unambiguous prefix for the alternative.

3. Optional Construct Choices are contained within braces, such as {description}. The user may select an optional construct in the exact same manner as a required choice. The user may pass-over an optional construct by pressing the return key.

STRUCTURE TRAVERSAL

The current position within the structure of a PSDL program may be changed by the following:

^N - next required construct.

^M or return - next construct, required or optional.

^P - previous required construct.

^H or backspace - previous construct, required or optional.

82

KEYBOARD INPUT
The current position while entering a particular text item
via keyboard input may be changed by the following:

DEL - Delete previous character.

^D - Delete current character.

^F - Move the cursor one character to the right.

^B - Move the cursor one character to the left.

SAVING THE PROTOTYPE
Once the prototype description has been entered the file may
be saved with the following method:

^X^W - a forms display appears at the bottom of the screen.

The cursor is positioned at file-name. Enter the name
psdl.text.

Press the tab key. The cursor is positioned at file-
type. Enter the type text .

Press the tab key again, and enter ESC s to execute the
command.

The file has been saved and the forms display is cleared
from the screen.

EXITING THE EDITOR
The editor is exited by typing ^C.

AUTHOR
Laura J. White

FILES

/caps/.syn_profile	default parameters for the psdl editor
/caps/syntax_editor/pev	terminal psdl editor
/caps/syntax_editor/pew	sunview psdl editor
/caps/syntax_editor/psdl.as.ssl	abstract syntax for psdl editor
/caps/syntax_editor/psdl.ad.ssl	attribute definitions for psdl editor
/caps/syntax_editor/psdl.ci.ssl	concrete input for psdl editor
/caps/syntax_editor/psdl.lex.ssl	lexical file for psdl editor

```
/caps/syntax_editor/psdl.tt.ssl        template transformations for psdl editor
/caps/syntax_editor/psdl.up.ssl        unparsing rules for psdl editor
```

SEE ALSO

White, L. J., The Development of a Rapid Prototyping Environment,
Master's Thesis, Naval Postgraduate School, Monterey, California,
December 1989.

BUGS

The text part of the PSDL grammar has not been implemented.
Identifier is currently used in place of text.

Some of the constructs remain at the current location,
instead of advancing forward after a selection is made. The
visibility behavior of optional constructs within some con-
structs is still "mysterious" in some instances. This must
be corrected before a sunview editor implementation will be
useful.

Name analysis and type checking is not performed in the
current version of pe.

A sample editing session for the construction of a simple prototype is described

within the rest of this section. We will show the contents of the video screen as the edit-

ing session progresses. Whenever a user presses a tab the command input prompt

appears at the top of the screen. The middle of the screen contains the program. The

current position in the abstract syntax tree is displayed on the bottom line to the left. If

command mode options are available, they appear to the right of the Positioned at

display. An option may be selected by entering an unambiguous prefix at the prompt or

passed over by pressing the return key. Figure 6-19 contains the contents of the video

screen once the syntax directed editor is started and initialized.

[component]

Positioned at psdl_components   type   operator

Figure 6-19. Sample Editing Session

We pressed the tab key to obtain the editor's command prompt. The command prompt appears at the top of the display as shown in Figure 6-20.

COMMAND:

[component]

Positioned at psdl_components   type   operator

Figure 6-20. Sample Editing Session

At this point, we entered the unambiguous prefix 'o' for an operator component. The command prompt disappears and the template for an operator component is displayed as shown in Figure 6-21.

```
OPERATOR  <identifier>
SPECIFICATION
END
[operator implementation]



Positioned at id
```

Figure 6-21. Sample Editing Session

We then entered the identifier name for the operator as shown in Figure 6-22. After the identifier name is entered, the placeholder for an optional interface construct appears. The choices for an optional interface construct are displayed on the bottom of the screen.

```
OPERATOR operator_1
SPECIFICATION
  (interface)
END
[operator implementation]



Positioned at optional_interface    input    output    states    generic
exceptions        timing_info
```

Figure 6-22. Sample Editing Session

We now chose to describe an input so we pressed the tab key and entered the unambiguous prefix 'i'. The display was updated with the input construct as shown in Figure 6-23.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    <identifier> : <identifier>
END
(operator implementation]


Positioned at id_list
```

Figure 6-23. Sample Editing Session

After we entered the identifier name of an input and its type as shown in Figure 6-24, the optional placeholder for a generic actual parameter is displayed.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer (generic actual parameters)
    (requirements)
END
(operator implementation]


Positioned at optional_generic_actuals   enter_generic_actual_parameters
```

Figure 6-24. Sample Editing Session

87

At this time we did not wish to specify any generic parameters so we pressed the return key to *pass over* this option. An optional placeholder for more_type_decls appears. We do not wish to specify any more inputs for our operator, nor any requirements for our input so we pressed the return key two more times. PSDL allows zero or more interface constructs so the interface option appears again as shown in Figure 6-25.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    (interface)
END
[operator implementation]
```

Positioned at optional_interface   input   output   states   generic
exceptions   timing_info

Figure 6-25. Sample Editing Session

We choose to enter timing information for our operator so we press the tab key, enter an unambiguous prefix 't' and press return. The placeholders for the optional timing constructs appear, and we are positioned at the first choice as shown in Figure 6-26.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    (met)
    (mcp)
    (mrt)
    (requirements)
END
[operator implementation]


Positioned at optional_met          enter_MET
```

Figure 6-26. Sample Editing Session

We chose to enter a MET so we pressed the tab key, entered an 'e' at the command

prompt and pressed return. The MET construct replaces the optional placeholder and we

are positio   integer as shown in Figure 6-27.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    MAXIMUM EXECUTION TIME <integer>
    (mcp)
    (mrt)
    (requirements)
END
[operator implementation]




Positioned at integer
```

Figure 6-27. Sample Editing Session

We entered a value for the time constraint and pressed return. The optional unit placeholder appeared as shown in Figure 6-28.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    MAXIMUM EXECUTION TIME 10 (units)
    (mcp)
    (mrt)
    (requirements)
END
(operator implementation)
```

Positioned at optional_unit   milliseconds   seconds   minutes   hours

Figure 6-28. Sample Editing Session

We selected milliseconds as our units by pressing the tab key then entering the unambiguous prefix 'mil'. After that we chose not to enter any more timing information or any requirements so we pressed return until we were given the opportunity to enter another interface construct. We chose not to enter any more of these so we pressed the return key again. We are now positioned at an optional keywords construct as shown in Figure 6-29.

90

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    MAXIMUM EXECUTION TIME 10 MS
    {keywords}
END
[operator implementation]
```

Positioned at optional_keywords   enter_keywords

Figure 6-29. Sample Editing Session

We chose not to enter any keywords so we pressed the return key. An optional description placeholder appeared which we passed over. After that, an optional axioms placeholder appeared which we also passed over. We were then positioned at an operator implementation as shown in Figure 6-30.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    MAXIMUM EXECUTION TIME 10 MS
END
[operator implementation]
```

Positioned at operator_impl   psdl_implementation   ada_implementation

Figure 6-30. Sample Editing Session

We selected a psdl implementation and the appropriate template for the implementation appeared as shown in Figure 6-31.

```
OPERATOR operator_1
SPECIFICATION
  INPUT
    x : integer
    MAXIMUM EXECUTION TIME 10 MS
END
IMPLEMENTATION
  GRAPH
    <identifier>.<identifier>-><identifier>
  (data stream)
  (timer)
  (control constraints)
  (description)
END

Positioned at id
```

Figure 6-31. Sample Editing Session

At this point we could continue to describe the implementation of our operator in the same manner as we did with the specification part, but we chose to exit the editor.

The PSDL editor uses a very small subset of the capabilities provided for an editor using the Cornell Synthesizer Generator. Simplicity and ease was a primary goal for our editor. This goal caused us to intentionally design our editor so that a very small subset of traversal commands would be required to use the editor effectively.

H. FUTURE WORK

The following items were either not completed in the initial draft of the editor due to time constraints or were identified as a result of testing.

1) The text rule in the PSDL grammar still requires implementation.

2) The concrete input syntax should be expanded so that a complete PSDL program may be parsed. An existing file can only be read into the syntax directed editor if concrete input syntax has been specified for the complete grammar.

3) Name analysis and type checking still require implementation.

4) The video version of the PSDL editor highlights the locator and the placeholder. The highlighted locator is a feature provided by the Cornell Synthesizer Generator. This feature is inconsistent with the editor we designed and confuses the user. The locator should not be visibly displayed.

5) The inconsistencies with placeholders must be corrected to provide a useful editor which provides a Sun version of the editor which utilizes the mouse, pop-up windows and scrolling windows. Once these inconsistencies are eliminated, the editor designer need only recompile the editor specifications with a SUN option flag or a X option flag in the Makefile to provide a graphic interface.

6) The current version of the editor has been integrated into CAPS in a very simple manner. More sophistication in terms of its interaction with the design database, software base and the graphic editor is necessary to meet the requirements of the current design.

VII. THE SOFTWARE DATABASE SYSTEM

The Software Database System consists of the following tools:

- the Design Database
- the Software Base
- the Software Design Management System

This system provides for the utilization of reusable prototype designs and reusable software components during the rapid prototyping process. These tools have not yet been integrated into CAPS. This is an important area which requires follow-on design and implementation. This chapter is included here for completeness in describing the development of our environment.

A survey of existing database management methodologies and systems has been conducted to determine the primary features required to support our software database system. Feasability studies have also been conducted to refine the requirements and interaction within CAPS using one selected database management system.

A. REUSABILITY

The key motivation for our software database system is reuse. The benefits and advantages of reusability include:

- improved software quality and maintenance
- increased programmer productivity and efficiency
- faster software development
- lower development costs

94

Galik [13] provides us with a brief synopsis of the following areas of research regarding software reusability:

- reusability and abstraction
- Ada and reusability
- the Ada software repository
- object-oriented programming and reusability
- a keyword based retrieval system
- a reusable software library
- classification and retrieval

Galik found that researchers generally agreed that a greater use of abstraction resulted in greater reusability. Parnas [30] notes that information hiding and abstraction generally produces software that is well-defined and well-documented, which tends towards reusability.

Reusability was a primary concern in the design of the Ada programming language, which supports both abstraction and information hiding. Ada separates the interface specification of a program unit from its implementation. These specifications determine how program units may be reused. Both the specification and the implementation parts may be reused. Ada provides a package subprogram which strongly supports the development of abstract data types. A key feature of Ada is the generic unit which is the primary mechanism provided in the language for building reusable software components.

The Ada Software Repository [31] is a public-domain collection of Ada programs and information. The software components are categorized by their high level applications. A programmer must scan an index, then browse each of the potential candidates for its suitability. This can be a very time-consuming process.

Johnson and Foote [32] state that object-oriented programming promotes software reuse. They state that class definitions provide modularity and information hiding which supports reuse. Class inheritance supports reuse by permitting classes to be modified to form subclasses.

Matsumoto [33] describes an object-oriented retrieval system being developed which is a simple keyword based system. A synonym library provides a standard normalized keyword for retrieval based on user input.

A Reusable Software Library (RSL) has been developed at Intermetrics, Inc. The RSL [34] is comprised of the RSL database and four subsystems which are:

- a library management subsystem
- a user query subsystem
- a software component retrieval and evaluation subsystem
- a software computer-aided design subsystem

This library classifies components by a set of attributes. Retrieval is an interactive process which identifies components which perform a desired function. The user assigns relative importance to the attributes of the components and the system evaluates and rates components based on the user's needs.

Prieto-Diaz and Freeman [35] state that the proper classification of reusable software components is the central issue in making reusability an attractive approach to software development. They propose an integrated classification scheme that is embedded within the retrieval system. An evaluation mechanism is provided to help users discriminate between similar components in the software base and to allow users to select components which will require minimal modifications. Their algorithm is shown in Figure 7-1. This algorithm provides the basis for our design in retrieving reusable components in CAPS.

```
begin
   search library
   if identical match then
      terminate
   else
      collect similar components
      for each componenet com
         compute degree of match
      end
      rank and select best
      modify component
   endif
end
```

Figure 7-1. Code Reuse Process

The classification scheme proposed by Prieto-Diaz utilizes a description of software

components with a sextuple containing attributes which capture the functional charac-

teristics of the component.

B. REQUIREMENTS

The requirements for our software database system were defined to provide the fol-

lowing capabilities:

- to store PSDL designs and software components
- to retrieve designs and software components for editing
- to retrieve designs and software components for review
- to delete designs and software components
- the CAPS user interface should interact with the software
  database system in such a way that the software management system
  is transparant to the protoype designer.

C. SURVEY OF DATABASE MANAGEMENT TECHNOLOGIES

Relational, Hierachical, Network and Object-Oriented database management systems were evaluated by Galik. He found that the first three did not support efficiency or productivity due to the following characteristics:

- fixed set of structures and operations
- limitations of a fixed set of predefined types
- redundancy of data
- lack of abstraction capabilities
- schema not easily modifiable

The object-oriented database management systems attempt to provide facilities which permit the definition of any structure or operation rather than a fixed set. Properties of an object-oriented database management system include:

- abstraction
- extensibility
- persistency
- active data

The object-oriented approach was selected as the most suitable technology for the implementation of our software database system. Within this type of system an object is considered as a description of an entity in some application domain. It may be a simple atomic representation or a composite structure. An object has a unique identifier and a set of operations which are defined for that object. Objects which are similar are grouped into classes. A class is also considered as an object. Class hierarchy and inheritance of properties permits the definition of subclasses which may inherit all properties of the class but may have additional local properties.

Vbase, a product of Ontologic Inc. was selected for CAPS. Vbase runs in the Unix environment on Sun workstations. The Vbase system consists of the following

components:

- the Vbase Database of persistent objects
- the Type Definition Language (TDL) provides the data model or conceptual schema for applications
- a C Object Processor which is the Vbase data manipulation language
- an Integrated Tool Set (ITS)
- an Object SQL which is the query facility for the retrival of objects

The process of defining a typical database design includes:

- define the objects
- define the properties of the objects
- define frequent operations performed on objects
- describe the objects and their properties in TDL
- compile and debug the TDL definitions
- develop COP routines which implement the defined operations
- compile and debug the COP programs

Galik and Douglas successfully implemented Vbase systems for the storage and retrieval of objects in the software base and the design database.

D. FUTURE INTEGRATION

We currently divide the software design management system into two COP components, one for the software base and one for the design database. The purpose of the design database and the interaction between the designer and the design database requires further refinement. This process has not been extended beyond the description provided by Douglas. We have extended and refined the design for the integration of the software database system from that previously described by Galik.

The software design management system for Ada components and the Vbase database of reusable Ada software components should reside in the directory /caps/software_base. The software design management system for PSDL prototype

99

specifications and the Vbase database of reusable prototype designs should reside in the directory /caps/design_database. The implementation defined filenames used in the feasibility studies, are suitable for the integration which we have currently defined. These files should exist as temporary data components with lifetimes based on their applicability.

The input file for the software base retrieval is currently a PSDL specification part and the output is an Ada component. The input file for a design database retrieval currently contains the desired properties of a PSDL description and the output is a complete PSDL description. We will describe the interaction of the software design management system in the context of constructing a prototype from within the graphic editor.

When a designer selects the graphic editor from the construction menu in the user interface, both the user interface process and the graphic editor process remain active. The designer may describe a prototype using a single operator. If the designer wishes to determine if a reusable Ada component already exists in the database for the operator just defined, then he selects the *search* button in the graphic editor. The search function should call the store function which will cause a file *NewNode.01* to be created which is a PSDL specification for the operator. The search function should make a copy of the PSDL specification into the file *PSDL_SPECS*. The search function should then execute the COP program which represents the database management system for the software base. If a match is not found in the software base, then the search function should remove the temporary file *PSDL_SPECS* and cause a message to be displayed in the message panel which states that a match was not found. If a match is found, then that reusable component now resides in the file *SB_OUT* and the search function should open a

popup window which overlays the top portion of the graphic editor. This window should not overlay the message panel or the drawing space. If a single match was found, then a statement which reflects that case is displayed and the designer is given an option to browse the component, edit the component, save the component or to quit the search process. If the designer chooses to browse the component, then the command *more SB_OUT* is automatically executed within the popup window. If the designer chooses to edit the component, then ideally an Ada syntax directed editor should automatically be executed on the file *SB_OUT* within the popup window. If the designer chooses to save the component then the modified component should be stored so that the designer may choose to discard it or add it to the software base during the *quit* process of the user interface. The component should also be appended to the file *SB_PACKAGE* which will be referenced by the translator during the *execute* process. If the designer chooses to quit the search process, then the files *PSDL_SPEC* and *SB_OUT* are removed and the pop-up window is closed.

The designer continues the construction and search processes until a complete PSDL description is represented by a tree which contains Ada components at all of its leaves.

101

VIII. THE TRANSLATOR

A. PREVIOUS DESIGN

The purpose of the translator is to produce an Ada translation of a PSDL prototype description. The translator performs lexical analysis of the internal textual representation of a PSDL system prototype, parses the prototype description and constructs an abstract syntax tree, and then evaluates the attributes of the tree to provide an Ada representation which utilizes PSDL abstract data types.

An Ada translation is a template with five major sections derived from the PSDL input program:

- exception declarations
- atomic operator driver headers
- atomic subprograms
- PSDL operator specification packages
- PSDL atomic operator driver subprograms

The exception declarations define the PSDL_EXCEPTION data type and all of the PSDL exceptions which may be raised in a PSDL program. The atomic operator driver headers are the interfaces for the subprogram names which will be called by the static and dynamic schedules. The atomic subprograms section contains all of the atomic level Ada code drawn from the reusable software base or entered by the designer during the construction of the prototype. The PSDL operator specification part contains Ada package specifications for all of the composite PSDL operator specifications. The PSDL atomic operator drivers are Ada subprograms which execute the atomic subprograms in

102

terms of the PSDL control constraints specified for each operator. The Ada package template is shown in Figure 8-1 [14].

```
package TL is

    type PSDL_EXCEPTION is (psdl_excp1, psdl_excp2,..., psdl_excpN;
    excp1, excp2,..., excpN : exception;


    procedure atomic_driver1;
    procedure atomic_driver2;
    ...
    procedure atomic_driverN;

end TL;

with PSDL_SYSTEM;
use PSDL_SYSTEM;

package body TL is

    atomic procedures drawn from the software base


    PSDL operator specification packages


    PSDL atomic operator driver procedures

end TL;
```

Figure 8-1. TL Package Template

103

The PSDL abstract data types implemented for the translator are:

- data streams
- state variables
- timers
- exceptions

There are two kinds of data streams in PSDL: sampled streams and data flow streams. A sampled stream is a data stream which has a persistent data value until it is overwritten with another value. When a value is read from a sampled stream, the value remains on the sampled stream. A data flow stream is a data stream which may only be written if the stream is empty and can only be read when a value exists on the data flow stream. Reading the value of the data flow stream consumes the value. Data streams have two defined error conditions, BUFFER_UNDERFLOW and BUFFER_OVERFLOW. The first error occurs if an attempt is made to write onto a data flow stream which has a value. The second error occurs if an attempt is made to read an uninitialized sampled stream or an empty data flow stream. The four different PSDL constructs which declare data streams are:

- input attribute
- output attribute
- states attribute
- streams

State variables are data streams which are automatically initialized.

A timer is a built-in data type which behaves as a simple digital stopwatch used to measure elapsed times. The operations available on timers include:

- start
- stop
- read
- reset

104

A timer may be represented by a state machine which has three states as shown in Figure 8-2 [14].
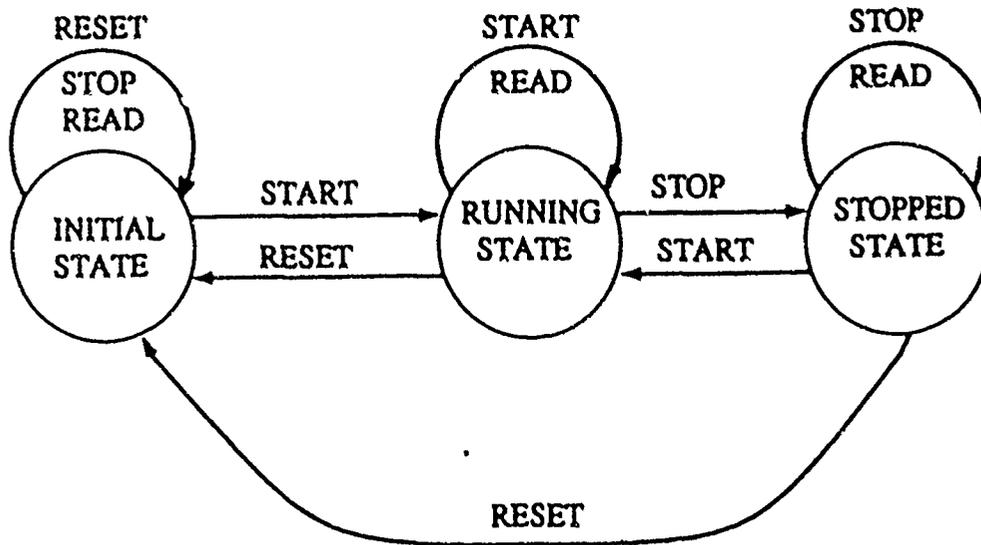
Figure 8-2. Timer State Machine

The read operation does not cause any change of state. A timer may be read at any time regardless of the current state. The value returned when reading a timer is always the amount of time that a timer has spent in the run ning state since the last transition from the initial state.

PSDL exceptions are special data types which may be written to any data stream without regard for the data stream's normal data type.

B. PREVIOUS IMPLEMENTATION

The translator is comprised of two main modules: the Kodiyak translator specification and the Ada package which implements the PSDL abstract data types.

The translator specification was implemented using the Kodiyak translator generator. Kodiyak is a UNIX based tool which was built on top of LEX and YACC. A Kodiyak program has three distinct parts. The first part provides the lexical rules for the source language. The second part provides the declarations for the attributes in the abstract syntax. The third part provides the attribute equations which are used to determine the values of the attributes in the abstract syntax tree. Four passes of the abstract syntax tree are performed. A pass is defined as a traversal from top to bottom or from bottom to top.

The first pass is a depth-first traversal of the tree from the root. Pass one collects the following information for each node in the tree:

- all operator names and their parent names
- all data stream names and whether they are sampled or data flow streams
- all exceptions declared in the program

The second pass traverses the tree from the leaves back to the root. The information determined for each node during pass one is synthesized up to the root. This information is collected in the Kodiyak data structure for a map. The global map contains the contextual data of the PSDL program.

The third pass routes the global map created in pass two back down the tree to start the translation process. Translation occurs at each node in the tree. The translations are to strings of Ada code. The leaves of the tree inherit the Ada translations.

106

The fourth pass collects the Ada translations from the leaves of the tree and constructs composition groups of translated Ada code. When the fourth pass reaches the root of the tree, then all the translation information is stored in the root.

C. MODIFICATIONS

We modified the design described by Altizer to improve the efficiency of the translation and compilation processes. This design modification has not been implemented at this time. The user interface was to insert the actual source code for reusable components which were used in the construction process into the Ada translation. It has since been realized that this should not be necessary. The reusable components should have source views and executable views. The designer may use the source views during the construction phase but then the executable views should be formed into a package which will be linked with the translator when the final executable prototype is created. The package of reusable components should have a unit name, such as SB_PACKAGE, and the translator must contain an Ada WITH SB_PACKAGE statement. This means that generic reusable components are instantiated and compiled when selected during the construction phase.

We also modified the implementation of the translator to require and recognize the END statement which brackets the PSDL IMPLEMENTATION ADA statement. This was necessary due to the design modification of the PSDL language which we described in chapter six.

D. INTEGRATION

The components which comprise the translator are contained in the directory /caps/translator. These files are:

translator.k   kodiyak translator specifications
psdl_system.a   Ada implementation of PSDL abstract data types

The data components which are used as input to the translator and the output produced by the translator are stored in the directory /caps/prototypes. These files are:

psdl.txt   input
tl.a   output

The Kodiyak Compiler resides in the subdirectory Kodiyak. An executable translator may be generated with *kc translator.k* in the subdirectory. The executable translator will have the file name *translator*. The executable translator must be moved to the parent directory.

The on-line manual page for Kodiyak is located at:

/n/suns2/usr/suns2/man/manl/kodiyak.1

The translator specifications and the Ada implementation of the PSDL data types are contained in Appendices M and N.

108

IX. THE STATIC SCHEDULER

A. PREVIOUS DESIGN

The purpose of the static scheduler is to schedule the PSDL operators in a prototype description so that the time constraints are satisfied during execution. If a feasible schedule exists, then the static scheduler creates an Ada program which controls the execution of these operators. A data flow diagram for the static scheduler is shown in Figure 9-1.
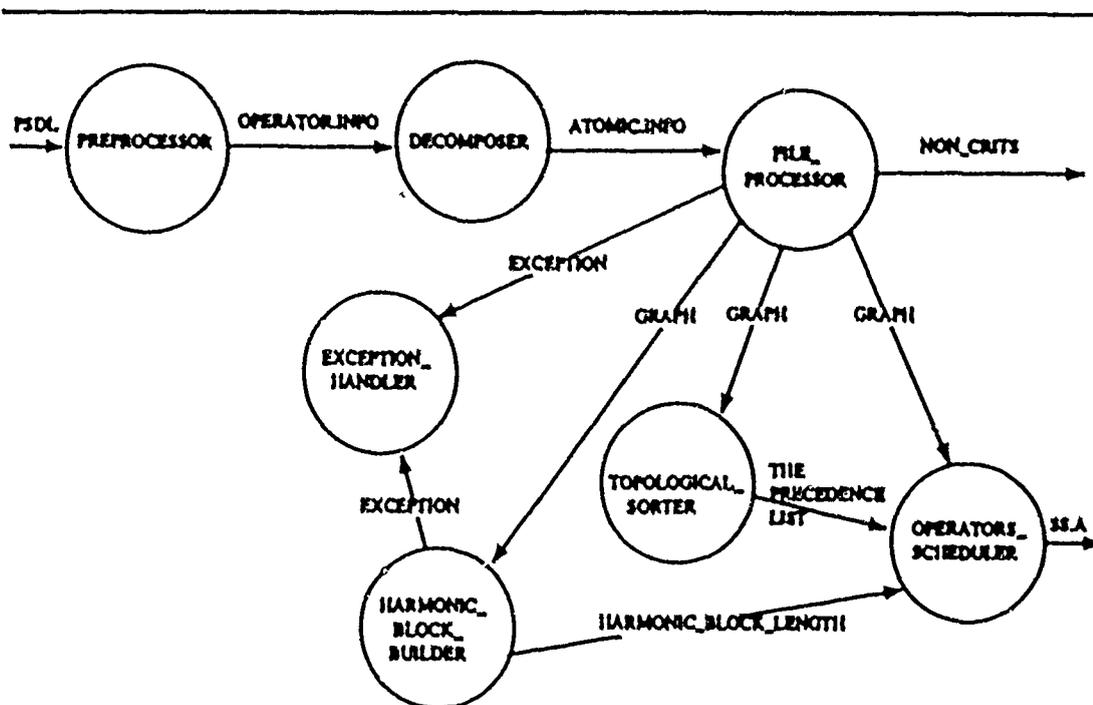
Figure 9-1. Static Scheduler

The Preprocessor reads an internal textual PSDL file and outputs a file which contains a list of all composite and atomic operators with their relevant characteristics such as their timing constraints and link information.

The Decomposer reads the output of the preprocessor and creates a file which only contains the atomic operators and their relevant characteristics.

The File_Processor reads the file of atomic operators and creates a graph structure which represents all of the atomic operators with critical timing constraints. It also creates a file which lists the atomic operators without critical timing constraints.

The Topological_Sorter uses the graph structure created by the File_Processor to build and output a precedence relationship that specifies which operators must complete their execution before other operators may start executing. This precedence relationship is always determined but is only used by some of the scheduling algorithms.

The Harmonic_Block_Builder calculates the periodic equivalents of the sporadic operators which have no defined periods. Then it checks if a harmonic block can be found for a single processor. If a harmonic block is found then it calculates and outputs a harmonic block length.

The Operators_Scheduler uses the graph structure, the precedence list and the harmonic block length to determine if a feasible schedule exists. If a schedule is feasible, one of six scheduling algorithms may be used to build a schedule.

The Exception_Handler manages the exceptions which pertain to critical operators scheduled by the static scheduler.

110

The static scheduler produces two outputs. One output is an Ada program which contains a schedule for all of the operators in a software system prototype with critical timing constraints. The other output is the file which contains the names of the atomic operators in the prototype which do not have any critical timing constraints.

B. IMPLEMENTATION

The static scheduler has been described and incrementally implemented by Marlowe [15] and Kilic [16]. The static scheduler preprocessor was implemented with Kodiyak specifications. The details of the Kodiyak Compiler were described in the previous chapter. The output of the preprocessor will be processed to produce an output which only contains atomic operators. An example of the atomic operators and their corresponding graph structure which would be created by the file processor are shown in Figure 9-2. There are currently three of six scheduling algorithms implemented in the static scheduler. The three algorithms which have been implemented are :

- the harmonic block with precedence constraints scheduling algorithm
- the earliest start scheduling algorithm
- the earliest deadline scheduling algorithm

The three scheduling algorithms which remain to be implemented are:

- the fixed priorities scheduling algorithm
- the minimize maximum tardiness with early start times scheduling algorithm
- the rate-monotonic priority assignment scheduling algorithm

These algorithms are explained in detail by Kilic. He implemented the static scheduler as a stand-alone tool which required modifications when integrated into CAPS. The primary capabilities of the two versions remain the same.

111
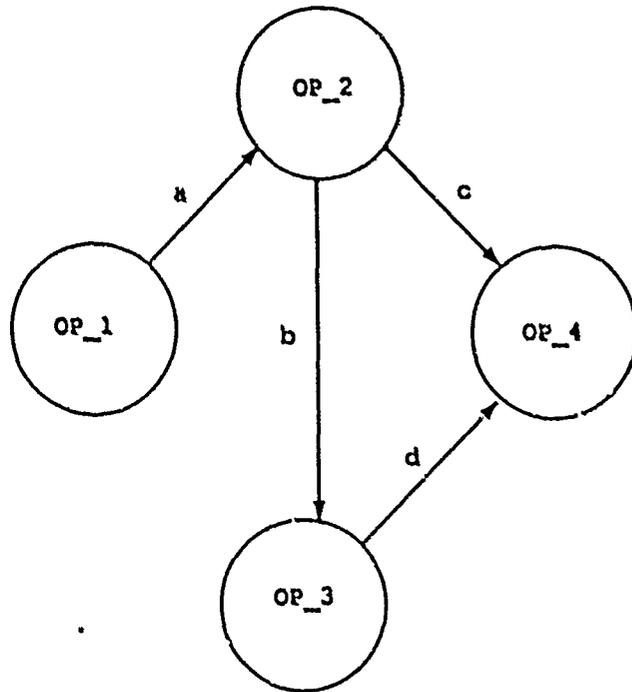
Figure 9-2. Decomposer Output And Graph Structure

C. MODIFICATIONS

The input to the static scheduler's preprocessor is a PSDL prototype description. The Kodiyak specifications for the preprocessor were modified to reflect the change in the PSDL grammar described in chapter six. An END statement is required to bracket an **IMPLEMENTATION ADA** statement.

The Ada implementations were modified to better serve the purposes of an integrated tool within CAPS. An algorithm selection menu and several of the notification messages were removed. The current design of the static scheduler in CAPS which interacts with a

112

designer applies the different scheduling algorithms in an implementation defined order until a solution is found or until all algorithms have failed to produce a schedule.

During the integration and testing of the static scheduler three major areas were identified for future modifications. The first is to develop some selection mechanism which matches the prototype design to an optimal scheduling algorithm. The selection of an algorithm will still remain transparent to the designer, but should become more efficient if the order of the algorithms are selected more intelligently. The second is that the decomposer module still remains to be implemented. Another area of future work which was originally identified by Marlowe is to define an algorithm and implementation which determines a harmonic block length for multiple processors.

D. INTEGRATION

The components which comprise the static scheduler are contained in the directory /caps/static_scheduler. These files are:

decomposer_b.a	- validates and decomposes output of preprocessor
decomposer_s.a	- validates and decomposes output of preprocessor
driver.a	- interface for stand-alone static scheduler
e_handler_b.a	- exception routines used by driver
e_handler_s.a	- exception routines used by driver
files.a	- global types and declarations for all ss programs
fp_b.a	- file processor
fp_s.a	- file processor
graphs_b.a	- generic type graph structure
graphs_s.a	- generic type graph structure
hbb_b.a	- harmonic block builder
hbb_s.a	- harmonic block builder
kc	- script to compile static scheduler preprocess pre_ss.k
pre_ss	- executable preprocessor
pre_ss.k	- kodiyak specifications for preprocessor
scheduler_b.a	- operators_scheduler
scheduler_s.a	- operators_scheduler

```
sequence_b.a          - generic type list structure
sequence_s.a          - generic type list structure
static_scheduler      - executable static_scheduler
t_sort_b.a            - topological sorter
t_sort_s.a            - topological sorter
```

The Kodiyak Compiler resides in the subdirectory /caps/static_scheduler/Kodiyak.

The preprocessor is generated by compiling with kc pre_ss.k in the Kodiyak subdirectory. The executable preprocessor pre_ss must be moved to the parent directory /caps/static_scheduler.

The user interface executes pre_ss with the command line equivalent:

pre_ss <file name> -o operator.info

The Ada components of the static_scheduler are compiled by:

a.make static_scheduler -f *.a -o static_scheduler
(where *.a uses all files listed above which have a .a suffix)

The static_scheduler is executed in the user interface by the command line equivalent:

static_scheduler

files.a is dependent upon:
    vstrings
    sequences
    graphs

decomposer_b.a, decomposer_s.a, e_handler_b.a, e_handler_s.a,
fp_b.a, fp_s.a, hbb_b.a, hbb_s.a, scheduler_b.a, scheduler_s.a,
t_sort_b.a, t_sort_s.a are all dependent upon:
    files (files.a)

driver.a is dependent upon
    decomposer (decomposer_b.a, decomposer_s.a)
    exception_handler (e_handler_b.a, e_handler_s.a)
    file_processor (fp_b.a, fp_s.a)
    harmonic_block_builder (hbb_b.a, hbb_s.a)
    operator_scheduler (scheduler_b.a, scheduler_s.a)
    topological_sorter (t_sort_b.a, t_sort_s.a)

114

pre_ss creates operator.info
decomposer reads operator.info and creates atomic.info
File_processor reads atomic.info
File_processor creates non_crits
Operator_scheduler creates ss.a

The components which comprise the static scheduler are contained in Appendices O

through AE.

X. THE DYNAMIC SCHEDULER

A. PREVIOUS DESIGN

The purpose of the dynamic scheduler was to coordinate the execution of all operators and their debuggers during the execution of a prototype. The static schedule, the non-time-critical operators and the debugging system were all components of the dynamic scheduler.

B. MODIFICATIONS

There was not any previous implementation of a dynamic scheduler. The previous design of the dynamic scheduler has been modified so that it only creates a schedule for the non-time-critical operators of a PSDL prototype description. A file which lists these operators is provided by the static scheduler. The dynamic scheduler is not concerned with the activities of any time-critical operators or with any special component which represents a debugging system. This modification still performs the functions required and provides for a simplified conceptual model of the execution support system. The dynamic schedule executes its operators in a sequential manner during the slack times between the execution of time-critical operators controlled by the static schedule. On a single processor each non-time-critical operator completes its execution before another non-time-critical operator is started.

C. INTEGRATION

The component which comprises the dynamic scheduler is located in the directory /caps/dynamic_scheduler. The file is dynamic_scheduler.a. The dynamic_scheduler is invoked by the user interface when the designer selects the *execute* option in the main menu. The dynamic scheduler reads the file *non.crits* which was produced by the static scheduler and creates a dynamic schedule which is placed in the file *ds.a*. The dynamic schedule, static schedule and Ada translation of the PSDL prototype are all compiled and linked into an executable data component which is currently called *prototype*. The dynamic scheduler is contained in Appendix AF.

XI. THE DEBUGGER

A. PREVIOUS DESIGN

The purpose of the debugger was to provide run-time support for the execution of
prototypes. The debugger was designed as two components: one for the static scheduler
and one for the dynamic scheduler. The debugger processed errors encountered by either
scheduler. The static debugger would process errors while attempting to create a
schedule and the dynamic debugger would process errors that occurred while the opera-
tors were executing. Both debuggers were to operate in a similar manner. They would
report an error condition, if possible correct the error, then permit the user to dictate
whether execution should continue or terminate. All information relating to an error
would also be written to a file for recall.

The static debugger was to process the following errors [17]:

- MET_Not Less_Than_MRT
- MET_Not_Less_Than_Period
- No_Initial_Link_Op
- No_Matches_Found
- MCP_NOT_Less_Than_MRT
- MET_Not_Less_Than_MCP
- No_Base_Block
- Fail_Half_Period
- Bad_Total_Time
- Ratio_Too_Big
- Over_Time
- Invalid_Schedule
- Schedule_Error
- MET_Required
- MET_GT_Parent
- MET_Sum_GT_Parent

- Crit_Op_Lacks_MET
- Crit_Op_Lacks_MET
- Excessive_Constraints_Altered

The dynamic debugger was intended to process errors identified during run-time execution of both time and non-time critical operators. These errors were [17]:

- Buffer_Underflow
- Buffer_Overflow
- Unprocessed_Exception
- Insufficient_MET
- Excessive_Execution

The dynamic debugger was to provide a user with an option to adjust the MET of an operator without terminating the execution of the prototype or to terminate the execution of the prototype.

B. PREVIOUS IMPLEMENTATION

The previous implementation consisted of two Ada programs. The implementation lacked the maturity required for integration into CAPS. The researchers involved with the parallel development of the static scheduler and the dynamic scheduler, Kilic and Palazzo, found the previous design and implementation of the debugger too awkward to integrate with their tools. The previous implementation has not been integrated into CAPS but should provide guidance in the design of the static and dynamic schedulers, and consideration in a more complete design of the debugger.

C. MODIFICATIONS

The design and implementation of the static debugger was essentially an exception handler and was partially implemented as a component of the static scheduler. The errors identified by Wood which were not implemented in the current static scheduler require further evaluation to determine their relevance.

The design and implementation of the dynamic debugger essentially provided an interface with the user to adjust the METs of operators which fail to satisfy their time constraints. The implementation did not contain any mechanism for actually effecting this change. The effect of changing the MET for one particular operator must be considered in terms of the static schedule. We have not currently determined whether or not a new static schedule should be built when a MET is changed in the run-time debugger. We also suspect that arbitrarily changing a MET in the run-time debugger may result in a prototype design which might not have a feasible static schedule. Possibly, the debugger should evaluate the side effects which will result in changing a MET and establish boundaries to the user. Care must be taken to ensure that the user is not allowed to modify the prototype design in the run-time debugger thus causing the prototype to *self-destruct*. These issues deserve careful consideration in the further design of a run-time debugger.

XII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

Our research has contributed towards the development of a rapid prototyping language and a rapid prototyping environment. The necessary characteristics, features, and difficulties with the development of such a language and environment have been refined as a result of the long term research effort which this thesis describes partially. The support towards software development and evolution which a rapid prototyping environment may or should provide and some of the important issues encountered with its development have also been better defined.

This thesis research encorporated the first attempt to assemble the previous work on individual tools into one integrated environment. This required a top-down view of the environment necessitating a much more comprehensive understanding of the major issues of a rapid prototyping environment than is required for a bottom-up view of a particular tool. As the integration progressed more and more knowledge of the design, structure and implementation of previously developed tools and knowledge of the technologies used in their implementation was required to enable the redesign, partial implementation and adaption into an integrated environment. Our current state of implementation reflects an ability to work in or understand PSDL, Ada, C, Pascal, Unix, Kodiyak, SSL and Sunview Graphics. Future modifications may also require this same knowledge.

121

The current state of development supports the view that it is impossible to completely state the requirements of a system in advance and that the development process itself changes the perceptions of what is possible or necessary. This has been reflected in previous chapters. The actual development of the tools and the environment has been performed primarily by many graduate students over a long period of time. Student contributions are constrained to relatively short time periods. Consequently, the design of CAPS has continued to evolve over time as different people have become involved. This work does not always reflect the ideal solution to a particular issue or implementation problem. It has been influenced by the early decision to permit students enough latitude to make the best use of their backgrounds and to permit them to explore the areas which interest them most. As the design and implementation have matured, the learning curve associated with an overall understanding of the environment has increased.

Related research areas currently in progress at the Naval Postgraduate School are:

- a graph model for software maintenance
- automatic merging of prototype versions
- generation of a CAPS interface with X windows
- prototype analysis and schedule matching
- survey and evaluation of current rapid prototyping processes

B. RECOMMENDATIONS

Version control for our implementations is currently performed manually. The previous lack of proper documentation and version control created significant difficulties during the initial stage of our integration. Previous incorrect versions of many of the implementations described in this thesis exist outside of the configuration used here. In

many cases, the implementations used in our integration are the only correct versions which exist on our system.

For evolution purposes implementation path names have been specified fully so that a new installation or a change in the system configuration may be effected by making global substitutions of the old system path names with the new system path names. System dependent or configuration dependent path names exist in the following files:

- caps.c
- graph.c
- nodes.p
- ge
- fp_b.a
- scheduler_b.a
- dynamic_scheduler.a

Significant follow-on research is required in the areas of view consistency, reusability, and run-time debugging. These issues and additional areas have been defined in greater detail within applicable chapters.

APPENDIX A PSDL GRAMMAR

Optional items are enclosed in [square brackets]. Items which may appear zero or
more times appear in { braces }. Terminal symbols appear in " double quotes ". Group-
ings appear in (parentheses).

```
psdl
        = {component}

component
        = data_type
        | operator

data_type
        = "type" id type_spec type_impl

operator
        = "operator" id operator_spec operator_impl

type_spec
        = "specification {type_decl} {"operator" id operator_spec}
          {functionality} "end"

type_impl
        = "implementation ada" id "(" text ")" "end"
        | "implementation" type_name {"operator" id operator_impl} "end"

operator_spec
        = "specification" {interface} {functionality} "end"

operator_impl
        = "implementation ada" id "(" text ")" "end"
        | "implementation" psdl_impl

type_decl
        = id_list ":" type_name {"," id_list ":" type_name}

functionality
        = [keywords] [informal_desc] [formal_desc]
```

```
psdl_impl
      = data_flow_diagram (streams) (timers) (control_constraints)
        (informal_desc) "end"

type_name
      = id
      | id "(" type_decl ")"

interface
      = attribute (reqmts_trace)

id_list
      = id ("," id)

keywords
      = "keywords" id_list

informal_desc
      = "description" "(" text ")"

formal_desc
      = "axioms" "(" text ")"

data_flow_diagram
      = "graph" link (link)

streams
      = "data stream" type_decl

timers
      = "timer" id_list

attribute
      = input
      | output
      | generic_param
      | states
      | exceptions
      | timing_info

input
      = "input" type_decl

output
      = "output" type_decl

generic_param
      = "generic" type_decl

states
      = "states" type_decl "initially" expression_list
```

```
exceptions
      ~ "exceptions" id_list

timing_info
      ~ ["maximum execution time" time] ["minimum calling period" time]
        ["maximum response time" time]

reqmts_trace
      ~ "by requirements" id_list

link
      ~ id "." id (":" time) "->" id

control_constraints
      ~ "control constraints" constraint (constraint)

constraint
      ~ "operator" id
        ("triggered" (trigger | (trigger) "if" predicate) (reqmts_trace))
        ("period" time (reqmts_trace))
        ("finish within" time (reqmts_trace))
        (constraint_options)

trigger
      ~ "by all" id_list
      | "by some" id_list

constraint_options
      ~ "output" id_list "if" predicate (reqmts_trace)
      | "exception" id ("if" predicate) (reqmts_trace)
      | timer_op id ("if" predicate) (reqmts_trace)

timer_op
      ~ "read timer"
      | "reset timer"
      | "start timer"
      | "stop timer"

expression_list
      ~ expression ("," expression)

time
      ~ integer (unit)

unit
      ~ "ms"
      | "sec"
      | "min"
      | "hours"
```

```
expression
      ~ constant
      | id
      | type_name "." id "(" expression_list ")"

predicate
      ~ simple_expression
      | simple_expression rel_op simple_expression

simple_expression
      ~ [sign] integer [unit]
      | [sign] real
      | ["not"] id
      | string
      | ["not"] "(" predicate ")"
      | ["not"] boolean_constant

bool_op
      ~ "and"
      | "or"

rel_op
      ~ "<"
      | "<="
      | ">"
      | ">="
      | "="
      | "/="
      | ":"

real
      ~ integer "." integer

integer
      ~ digit (digit)

boolean_constant
      ~ "true"
      | "false"

numeric_constant
      ~ real
      | integer

constant
      ~ numberic_constant
      | boolean_constant
```

127

```
sign
      = "+"
      | "-"

char
      = any printable character except ")"

digit
      = "0 .. 9"

letter
      = "a .. z"
      | "A .. Z"
      | "_"

alpha_numberic
      = letter
      | digit

id
      = letter (alpha_numeric)

string
      = """ (char) """

text
      = (char)
```

APPENDIX B USER INTERFACE

```
/******************************************************************
FILE:  caps.c
AUTHOR: Laura J. White
DATE:   29 December 89
PURPOSE: CAPS - User Interface
******************************************************************/
#include <stdio.h>
#include <sys/file.h>
#include <sys/wait.h>
#include <signal.h>

/* system dependent pathnames */
#define SERVER            "suns2"
#define SHELL             "/bin/csh"
#define R_SHELL           "/n/suns2/usr/ucb/rsh"
#define TEXT_FILE         "/n/suns2/work/caps/prototypes/psdl.txt"
#define ADA_COMPILER      "/n/suns2/usr/suns2/VADS/bin/a.make"
#define GRAPHIC_EDITOR    "/n/suns2/work/caps/graphic_editor/ge"
#define SDE               "/n/suns2/work/caps/syntax_editor/pev"
#define LOCAL_MANUAL      "man pe"
#define REMOTE_MANUAL     "rsh suns2 man pe"
#define TRANSLATOR        "/n/suns2/work/caps/translator/translator"
#define DYNAMIC           "/n/suns2/work/caps/dynamic_scheduler/dynamic_scheduler"
#define STATIC            "/n/suns2/work/caps/static_scheduler/static_scheduler"
#define PRE_SS            "/n/suns2/work/caps/static_scheduler/pre_ss"
#define PSDL_COMPOSITES   "/n/suns2/work/caps/prototypes/operator.info"
#define PSDL_ATOMICS      "/n/suns2/work/caps/prototypes/atomic.info"
#define PROTOTYPE         "/n/suns2/work/caps/prototypes/prototype"
#define MV_TL_WORK        "mv /n/suns2/work/caps/prototypes/tl.a \
                              /n/suns2/work/caps/tl.a"
#define MV_TL_HOME        "mv /n/suns2/work/caps/tl.a \
                              /n/suns2/work/caps/prototypes/tl.a"
#define MV_SS_WORK        "mv /n/suns2/work/caps/prototypes/ss.a \
                              /n/suns2/work/caps/ss.a"
#define MV_SS_HOME        "mv /n/suns2/work/caps/ss.a \
                              /n/suns2/work/caps/prototypes/ss.a"
#define MV_DS_WORK        "mv /n/suns2/work/caps/prototypes/ds.a \
                              /n/suns2/work/caps/ds.a"
#define MV_DS_HOME        "mv /n/suns2/work/caps/ds.a \
                              /n/suns2/work/caps/prototypes/ds.a"


union wait status;
int   code;
```

```
main()
/********************************************************************
Function:       Main menu for CAPS
Date:           9 May 89
Called By:      command line
Calls:          construct(), execute(), modify()
Side Effects:   int code
********************************************************************/
{
    char choice;
    char carriage_return;
    int  caps_done = 0;

    while(!caps_done) {
        system("clear");
        fprintf(stdout,"\n\n\n\n\n\n");
        fprintf(stdout,"      COMPUTER AIDED PROTOTYPING SYSTEM\n");
        fprintf(stdout,"\n\n");
        fprintf(stdout,"                    (c)onstruct\n");
        fprintf(stdout,"                    (e)xecute\n");
        fprintf(stdout,"                    (m)odify\n");
        fprintf(stdout,"                    (q)uit\n");
        fprintf(stdout,"\n");
        fprintf(stdout,"        Select Option: ");
        fscanf(stdin,"%c",&choice);
        fscanf(stdin,"%c",&carriage_return);
        switch(choice) {
            case 'c':
                construct();
                break;
            case 'e':
                execute();
                break;
            case 'm':
                modify();
                break;
            case 'q':
                caps_done++;
                break;
            default:
                fprintf(stdout,"\n        Invalid Choice\n");
                sleep(3);
                break;
        }
    }
    system("clear");
    exit(0);
}
```

130

```c
int construct()
/********************************************************************
Function:        construct selection from caps main menu
                 submenu for graphic_editor or syntax directed editor
Date:            6 May 1989
Called By:       main()
Calls:           file /caps/graphic_editor/ge
                 syntax_editor()
Side Effects:    int   code
                 file  /caps/prototypes/psdl.txt
                 file  /caps/prototypes/graph.pic
                 file  /caps/prototypes/graph.links
                 file  /caps/prototypes/psdl.ds
                 files /caps/prototypes/NewNode.XX
********************************************************************/
{
    char editor;
    char carriage_return;
    int  construct_done = 0;

    while(!construct_done) {
        system("clear");
        fprintf(stdout,"\n\n\n\n        CONSTRUCT MODE\n\n");
        fprintf(stdout,"                (g)raphic editor\n");
        fprintf(stdout,"                (s)yntax directed editor\n");
        fprintf(stdout,"                (r)eturn to main menu\n");
        fprintf(stdout,"\n     Select Option: ");
        fscanf(stdin,"%c",&editor);
        fscanf(stdin,"%c",&carriage_return);
        switch(editor) {
            case 'g':
                system("clear");
                if (fork() == 0) {
                    code=execl(SHELL,SHELL,"-f",GRAPHIC_EDITOR,0);
                    exit(code);
                }
                wait(&status);
                break;
            case 's':
                syntax_editor();
                break;
            case 'r':
                construct_done++;
                break;
            default:
                fprintf(stdout,"\n        Invalid Choice\n");
                sleep(3);
                break;
        }
    }
}
```

```
int execute()
/*****************************************************************************
Function:       execute selection from caps main menu
Date:           29 Dec 89
Called By:      main()
Calls:          program /caps/translator/translator
                program /caps/static_scheduler/pre_ss
                program /caps/static_scheduler/static_scheduler
                program /caps/dynamic_scheduler/dynamic_scheduler
Side Effects:   file /caps/prototypes/psdl_tl.a
                file /caps/prototypes/operator.info
                file /caps/prototypes/ss.a
                file /caps/prototypes/non_crits
                file /caps/prototypes/ds.a
*****************************************************************************/
{
    char hostname[32];
    int check_prototype;

    system("clear");

    /* check for source file from the graphic editor or syntax directed
       editor which represents current prototype design */
    if ((check_prototype = open(TEXT_FILE, O_RDONLY,0)) == -1) {
        fprintf(stdout,"\nNo Completed Prototype Available\n");
        sleep(3);
        return 0;
    }
    close(check_prototype);

    fprintf(stdout,"\nTranslating ...\n");
    if (fork() == 0) {
        code=execl(TRANSLATOR, TRANSLATOR, TEXT_FILE, "-o",
                PSDL_TL_HOME, 0);
        exit(code);
    }
    wait(&status);

    fprintf(stdout,"\nBuilding Static Schedule ...\n");
    if (fork() == 0) {
        code=execl(PRE_SS, PRE_SS, TEXT_FILE, "-o",
                PSDL_COMPOSITES, 0);
        exit(code);
    }
    wait(&status);

    if (fork() == 0) {
        code=execl(STATIC, STATIC, 0);
        exit(code);
    }
    wait(&status);
```

132

```c
fprintf(stdout,"\nBuilding Dynamic Schedule ...\n");
if (fork() == 0) {
    code=execl(DYNAMIC, DYNAMIC, 0);
    exit(code);
}
wait(&status);

system(MV_TL_WORK);
system(MV_DS_WORK);
system(MV_SS_WORK);

gethostname(hostname,sizeof(hostname));
if (!strcmp(hostname,SERVER)) {
    if (fork() == 0) {
        code=execl(ADA_COMPILER, ADA_COMPILER,
                "static_schedule", "-f", "tl.a", "ds.a", "ss.a",
                "-o", PROTOTYPE, 0);
        exit(code);
    }
}
else {
    if (fork() == 0) {
        code=execl(R_SHELL, R_SHELL, SERVER, "a.make",
                "static_schedule", "-f", "tl.a", "ds.a", "ss.a",
                "-o", PROTOTYPE, 0);
        exit(code);
    }
}
fprintf(stdout,"\nCompiling ...\n");
wait(&status);

system(MV_TL_HOME);
system(MV_DS_HOME);
system(MV_SS_HOME);

/* execute */
signal(SIGINT, SIG_IGN);
fprintf(stdout,"\nExecuting ...\n");
sleep(1);
system("clear");
if (fork() == 0) {
    signal(SIGINT,SIG_DFL);
    code=execl(PROTOTYPE, PROTOTYPE, 0);
    exit(code);
}
wait(&status);
fprintf(stdout,"\nExecution Completed\n");
sleep(3);
signal(SIGINT,SIG_DFL);
}
```

```
int modify()
/*******************************************************************************
Function:       modify selection from caps main menu
Date:           6 May 89
Called By:      main()
Calls:          none
Side Effects:   none
*******************************************************************************/
{
    fprintf(stdout,"\n       Modifications in progress\n");
    sleep(3);
}
```

```
int syntax_editor()
/******************************************************************
Function:        syntax directed editor option from construct submenu
Date:            6 May 89
Called By:       construct()
Calls:           none
Side Effects:    file /caps/prototypes/psdl.txt
******************************************************************/
{
    char text;
    char hostname[32];
    char carriage_return;

    system("clear");
    fprintf(stdout,"\n\n\n        SYNTAX DIRECTED EDITOR\n\n\n");
    fprintf(stdout,"            Do you desire instructions\n");
    fprintf(stdout,"                (y)es\n");
    fprintf(stdout,"                (n)o\n");
    fprintf(stdout,"\n");
    fprintf(stdout,"        Select Option: ");
    fscanf(stdin,"%c",&text);
    fscanf(stdin,"%c",&carriage_return);
    if (text == 'y') {
        system("clear");
        gethostname(hostname,sizeof(hostname));
        if (!strcmp(hostname,SERVER))
            system(LOCAL_MANUAL);
        else
            system(REMOTE_MANUAL);
    }
    if (text == 'y') {
        fprintf(stdout,"\n\nPress Carriage Return to Continue : ");
        fscanf(stdin,"%c",&carriage_return);
    }
    system("clear");
    if (fork() == 0) {
        code=execl(SDE, SDE, 0);
        exit(code);
    }
    wait(&status);
    system("mv psdl.txt /n/suns2/work/caps/prototypes");
}
```

APPENDIX C SHELL SCRIPT FOR GRAPHIC EDITOR

```
#--------------------------------------------------------------------
# File:    ge
# Purpose: shell to run the graphic editor - directs input and output
# Author:  laura . white
# Date:    17 dec 1989
#--------------------------------------------------------------------

/n/suns2/work/caps/graphic_editor/graph
/n/suns2/work/caps/graphic_editor/nodes < /n/suns2/work/caps/prototypes\
/graph.links
cat /n/suns2/work/caps/prototypes/graph.links /n/suns2/work/caps\
/prototypes/psdl.ds >> /n/suns2/work/caps/prototypes/psdl.imp
```

APPENDIX D GRAPHIC EDITOR

```
/*********************************************************************
  PROGRAM:    GRAPH.C
  AUTHORS:    ROGER K. THORSTEN
              LAURA J. WHITE
  DATE:       10 November 1988
 **********************************************************************/

/* compile this program with "makid graph.c" */
#ifdef MAKID
static char
*makid[] = {
"@(M)cc -g graph.c -o graph -lm -lsuntool -lsunwindow -lpixrect"
};
#endif

#include <stdio.h>
#include <suntool/sunview.h>
#include <suntool/canvas.h>
#include <suntool/panel.h>
#include <suntool/seln.h>
#include <math.h>
#include <string.h>
#include <ctype.h>

/* system dependent names */
#define ICON        "/n/suns2/work/caps/graphic_editor/editor.icon"
#define PRINT       "screendump | rsh virgo lpr -Pssl -v "
#define SERVER1     "virgo"
#define SERVER2     "suns2"
#define SERVER3     "libra"
#define SERVER4     "taurus"

/* define constants for the editing modes */
#define OPERATOR    0
#define DATA_FLOW   1
#define SELF_LOOP   2
#define INPUT       3
#define OUTPUT      4
#define EXTERNAL    5

#define PI 3.141592654

#define DISP_WIDTH 142                          /* the display width */
#define DISP_HT     55                          /* the display height */
```

```
#define ARROW_LENGTH 9                              /* length of the arrow head */
#define TEXT_MAX_LEN 35                        /* length of name which is visible */
#define TIME_MAX_LEN 10                     /* length of the time which is visible */

#define PROXIMITY  25

                                             /* import predefined editor icon */
static short editor_icon[] = {
#include ICON
};


DEFINE_ICON_FROM_IMAGE(editor, editor_icon);


Frame    frame;                             /* define the handle for the frame */

Panel    mouse_panel,      /* defines the handle for the mouse interface panel */
         op_mode_panel,  /* define the handle for the op mode selection panel */
         edit_mode_panel,           /* define the handle for the side panel */
         time_panel,                /* define the handle for the time panel */
         name_panel,        /* define the handle for the name reading panel */
         message_panel;     /* define the handle for the message panel*/

Canvas   drawing_canvas;         /* define the handle for the drawing canvas */

Event    *event;                           /* define the handle for events */

Pixfont  *bold;                       /* define the handle for the borders */

Pixwin   *drawing_pw;          /* define the handle for the drawing pixwin

int      server = 0;         /* gloabal - flag for server/diskless sun    */

int      edit_mode;            /* global - stores the current edit mode */

int      name_checked = 0,         /* global - signals that name is valid */
         time_checked = 0;         /* global - signals that time is valid */

int      graph_saved = 0;           /* global - signals whether or not
                                              the graph has been saved */

Panel_item object_name,                          /* handle for the name */
           message,                              /* handle for the msg  */
           time_constraint;                      /* handle for the time */

char     *tmp_buf,                   /* global - buffer to read the name into */
         *tmp_buf1;      /* global - buffer to read the time constraint into */

FILE     *f,                         /* define the PSDL link statement file */
         *g;

typedef struct {
    int   length;                    /* the number of characters in the name */
```

138

```c
        char  string[80];                          /* array to hold the name or time string */
}Name,Time;

typedef struct line{                               /* stores output and data flow lines */
    Name  *name;                                   /* name of line */
    int   lntype;                                  /* identifies the type of line it is */
    int   xstart;                                  /* the x coord of the lines starting posit */
    int   ystart;                                  /* the y coord of the lines starting posit */
    int   xstop;                                   /* x coord of its stopping posit */
    int   ystop;                                   /* y coord of its stopping posit */
    Name  *dest;                                   /* operator that the line terminates at */
    struct line  *next;                            /* pointer to the next line from this operator */
}Line;

typedef struct operator{        /* storage stucture for the operators and inputs */
    Name  *name;                                   /* operator's name */
    int   optype;               /* identifies contents as an operator or external */
    int   xstart;               /* x coord where operator should start to be drawn */
    int   ystart;               /* y coord where operator should start to be drawn */
    int   xstop;                            /* x coord of the operator's opposite corner */
    int   ystop;                            /* y coord of the operator's opposite corner */
    Time  *time_const;                      /* maximum execution time for the operator */
    Line  *head;                                   /* head of the operator's output list */
    Line  *tail;                                   /* tail of the operator's output list */
    struct operator *next;                         /* pointer to next operator in list */
}Operator;

typedef struct{                                /* the list for operators and inputs */
    Operator *head;                            /* pointer to the head of the list */
    Operator *tail;                            /* pointer to the tail of the list */
}Operator_list;

Operator_list operator_list;
Operator_list *op_list = &operator_list;
Operator        *op, *sop, *dop;


Name            name_pointer;
Name            *name = &name_pointer;
Time            *tc;
Line            *ln;


/* forward declarations of functions */
static   Notify_value process_canvas_events();
static   Notify_value mode_select();

Operator *alloc_operator();
Operator *pick_operator();
Operator *create_op();

Line     *alloc_line();
Line     *pick_line();
```

139

```
Line      *create_line();

Name      *external(),
Name      *get_name();

Time      *get_time_const();

int       is_op_pick();
int       is_line_pick();
int       is_valid_ada_id();
int       is_valid_time_const();

int append_to_op_list()
int get_hostname();
int quit_proc();
int load_proc();
int dump_screen();
int search();
int compose();
int decompose();
int store_proc();
int out_of_mem();
int input_text();
int input_time();
int append_line_to_op();
int is_op_pick();
int is_input_pick();
int process_operator();
int process_line();
int rubber_band();
int redraw_diagram();
int delete_line();
int delete_op();
int delete_input_lines();
int display_error_msg();
int display_name();
int display_tc();
int draw_arrowhead();
int draw_object();
int create_PSDL();
int store_diagram();
```

```c
main(argc, argv)
    int    argc;
    char **argv;
/*********************************************************************
function:    Sets up graphic editor
called by:   CAPS User Interface
calls:       create_mouse_panel()
             create_operating_mode_panel()
             create_editing_mode_panel()
             create_name_panel()
             create_time_panel()
             create_message_panel()
*********************************************************************/
{
    get_hostname();

    /* cause borders to highlight if region entered */
    bold = pf_open("/usr/lib/fonts/fixedwidthfonts/screen.b.12");
    if (bold == NULL) exit(1);

    /* create the outer display frame */
    frame = window_create(NULL, FRAME,
                FRAME_LABEL,         "CAPS - GRAPHIC EDITOR",
                FRAME_ICON,          &editor,
                FRAME_ARGS,          argc, argv,
                WIN_ERROR_MSG,       "can't create window.",
                WIN_X,               2,
                WIN_Y,               1,
                WIN_ROWS,            DISP_HT,
                WIN_COLUMNS,         DISP_WIDTH,
                0);

    /* create mouse_panel */
    mouse_panel = window_create(frame, PANEL, WIN_FONT , bold, 0);
    create_mouse_panel();

    /* create op_mode_panel */
    op_mode_panel = window_create(frame, PANEL, WIN_FONT , bold, 0);
    create_operating_mode_panel();

    /* create editing mode panel */
    edit_mode_panel = window_create(frame, PANEL, WIN_FONT, bold, 0);
    create_editing_mode_panel();

    /* create panel to read object names */
    name_panel = window_create(frame, PANEL, WIN_FONT, bold, 0);
    create_name_panel();

    /* create panel to read operator time constraints */
    time_panel = window_create(frame, PANEL, WIN_FONT, bold, 0);
    create_time_panel();
```

```
/* create message_panel */
message_panel = window_create(frame, PANEL, WIN_FONT, bold, 0);
create_message_panel();

/* create canvas to draw on */
drawing_canvas = window_create(frame, CANVAS, WIN_FONT, bold,
                    WIN_CONSUME_KBD_EVENT, WIN_ASCII_EVENTS,
                    WIN_EVENT_PROC, process_canvas_events,
                    CANVAS_RETAINED, TRUE, 0);

/* cause drag events to be accepted */
window_set(drawing_canvas,WIN_CONSUME_PICK_EVENT, LOC_DRAG, 0);
drawing_pw = canvas_pixwin(drawing_canvas);

/* initialize the operator list */
operator_list.head = operator_list.tail = NULL;

/* poll for events in the frame */
window_main_loop(frame);

}
```

```
get_hostname()
/*******************************************************************************
 function:    This function determines if the user is using a sun server.
              The screendump capability for the graphic editor is only functional
              if the editor is operating on a sun server.  The control panel
              button and registration of the dump_screen function are not
              part of the graphic display if using a diskless workstation.
 called by:  main()
 calls:      none

 *******************************************************************************/
{
    char hostname[32];

    gethostname(hostname,sizeof(hostname));
    if (!strcmp(hostname,SERVER1))
        server++;
    if (!strcmp(hostname,SERVER2))
        server++;
    if (!strcmp(hostname,SERVER3))
        server++;
    if (!strcmp(hostname,SERVER4))
        server++;
}
```

143

```
create_mouse_panel()
/*****************************************************************************
  function:   Draws the mouse interface panel which contains messages which
              describe the functionality of the mouse buttons
  called by: main()
  calls:      none
*****************************************************************************/
{
    /* display panel messages */
    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    " MOUSE INTERFACE:", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "                                                    ", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "                                                    ", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "        Left Mouse   SELECTS graphic editor functions and", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "locations for new graphic objects", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "                                              ", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "        Middle Mouse MOVES graphic objects", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "                                                    ", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "                                    ", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "        Right Mouse  DELETES when positioned:", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "within an operator, on the tail of a self loop,", 0);

    panel_create_item(mouse_panel, PANEL_MESSAGE, PANEL_LABEL_STRING,
    "on the tail/head of a data flow, input or output", 0);

    /* fit border around the mouse panel */
    window_fit_height(mouse_panel);
}
```

144

```
create_operating_mode_panel()
/************************************************************************
 function:  This procedure builds the operating mode panel for the graphic
            editor, which consists of the buttons :  "Print Display" (if using
            a server), "Load Existing", "Store", and "Quit".
 called by: main()
 calls:     none
 ************************************************************************/
(
    /* display panel message */
    panel_create_item(op_mode_panel, PANEL_MESSAGE,
        PANEL_LABEL_STRING," OPERATING MODE:", 0);


    /* create button to permit a screendump of display */
    if (server) (
        panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
        panel_button_image(op_mode_panel, "Print Design", 0,          0),
        PANEL_NOTIFY_PROC, dump_screen, 0);
    )


    /* create button to cause data to be read from the data base */
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
        panel_button_image(op_mode_panel, "Load Existing", 0,          0),
        PANEL_NOTIFY_PROC, load_proc, 0);


    /* create button to interface with databases */
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
        panel_button_image(op_mode_panel, "Search", 0,          0),
        PANEL_NOTIFY_PROC, search, 0);


    /* create button to compose design*/
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
        panel_button_image(op_mode_panel, "Compose", 0,          0),
        PANEL_NOTIFY_P )C, compose, 0);


    /* create button to decompose design*/
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE,
        panel_button_image(op_mode_panel, "Decompose", 0,          0),
        PANEL_NOTIFY_PROC, decompose, 0);


    /* create button to store the diagram in the data base */
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE, panel_button_image(op_mode_panel, "Store", 0, 0),
        PANEL_NOTIFY_PROC, store_proc, 0);
```

145

```
    /* create button to terminate the program */
    panel_create_item(op_mode_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE, panel_button_image(op_mode_panel, "Quit", 0, 0),
        PANEL_NOTIFY_PROC, quit_proc, 0);

    /* fit border around the top panel */
    window_fit_height(op_mode_panel);
}
```

```
create_editing_mode_panel()
/************************************************************************
 function:  builds the editing mode panel for the graphic
            editor
 called by: main()
 calls:     none
 ***********************************************************************/
{
    /* create the mode select panel */
    panel_create_item(edit_mode_panel, PANEL_CHOICE,
            PANEL_LABEL_STRING,     " EDITING MODE:  ",
            PANEL_CHOICE_STRINGS,   "     Draw Operator    ",
                                    "     Draw Data Flow    ",
                                    "     Draw Self Loop    ",
                                    "       Draw Input      ",
                                    "       Draw Output     ",
                                    0,
            PANEL_FEEDBACK,         PANEL_INVERTED,
            PANEL_NOTIFY_PROC,      mode_select, 0);

    /* fit window around the editing mode panel */
    window_fit_height(edit_mode_panel);
}
```

```
create_name_panel()
/***************************************************************************
 function:  builds the identifier name panel for the graphic editor
 called by: main()
 calls:     none
 ***************************************************************************/
{
    object_name = panel_create_item(name_panel, PANEL_TEXT,
        PANEL_LABEL_STRING,          " IDENTIFIER NAME:",
        PANEL_VALUE,                 "",
        PANEL_VALUE_DISPLAY_LENGTH, TEXT_MAX_LEN,
        0);

    panel_create_item(name_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE, panel_button_image(name_panel, "Read Name", 0,0),
        PANEL_NOTIFY_PROC, input_text,
        0);

    window_fit_height(name_panel);
}
```

```
create_time_panel()
/******************************************************************************
 function:  builds the time constraint panel for the graphic editor
 called by: main()
 calls:     none
 ******************************************************************************/
{
    time_constraint = panel_create_item(time_panel, PANEL_TEXT,
        PANEL_LABEL_STRING,         " TIME CONSTRAINT:",
        PANEL_VALUE,                "",
        PANEL_VALUE_DISPLAY_LENGTH,TIME_MAX_LEN,
        0);

    panel_create_item(time_panel, PANEL_MESSAGE,
        PANEL_LABEL_STRING,  "                        ",
        0);

    panel_create_item(time_panel, PANEL_BUTTON,
        PANEL_LABEL_IMAGE, panel_button_image(time_panel, "Read Time", 0,0),
        PANEL_NOTIFY_PROC, input_time,
        0);

    window_fit_height(time_panel);
}
```

```
create_message_panel()
/*********************************************************************************
 function:  builds message panel for editor error messages to the user
 called by: main()
 calls:     none
 ********************************************************************************/
{
    /* display messages panel label*/
    message=panel_create_item(message_panel, PANEL_MESSAGE,
            PANEL_LABEL_STRING,
            " MESSAGE PANEL:", 0);

    window_fit_height(message_panel);
}
```

```
static Notify_value
mode_select(item, value, event)
    Panel_item item;
    int      value;
    Event    *event;
/***************************************************************************
 function:  sets the mode that the editor is operating in by setting the
            global variable "edit_mode" to one of the predefined mode constants
 called by: notifier
 calls:     none
 ***************************************************************************/
{
    switch(value) {
        case OPERATOR:
                        edit_mode = OPERATOR;
                        break;
        case DATA_FLOW:
                        edit_mode = DATA_FLOW;
                        break;
        case SELF_LOOP:
                        edit_mode = SELF_LOOP;
                        break;
        case INPUT:
                        edit_mode = INPUT;
                        break;
        case OUTPUT:
                        edit_mode = OUTPUT;
                        break;
        default:
                        break;
    }
    return;
}
```

```
quit_proc()
/*********************************************************************************
  function:  sets the mode that the editor is operating in by setting the
             global variable "edit_mode" to one of the predefined mode constants
  called by: notifier
  calls:     display_error_msg()
 *********************************************************************************/
{
    if (graph_saved) {
        window_set(frame, FRAME_NO_CONFIRM, TRUE, 0);
        window_destroy(frame);
    }
    else {
        display_error_msg(6);
        window_set(frame, 0);
        window_destroy(frame);
        display_error_msg(1);
    }
}
```

152

```
load_proc()
/**********************************************************************

function:     This function causes a previously drawn and stored diagram to
              be loaded when the load_existing button is selected.
called by:    notifier
calls:        create_op()
              append_to_op_list()
              create_line()
              append_line_to_op()
**********************************************************************/
{
    int   optype,x1,y1,x2,y2;
    Name *oname,
         *dest;
    Time *tc;

    graph_saved = 0;
    g = fopen("/n/suns2/work/caps/prototypes/graph.pic","r");

    while (!feof(g)) {
        oname = (Name *)malloc(sizeof(Name));
        fscanf(g,"%d\n",&optype);
        fscanf(g,"%d\n",&x1);
        fscanf(g,"%d\n",&y1);
        fscanf(g,"%d\n",&x2);
        fscanf(g,"%d\n",&y2);
        fscanf(g,"%s\n",oname->string);
        oname->length = strlen(oname->string);
        if ((optype == OPERATOR) || (optype == EXTERNAL)) {
            tc = (Time *)malloc(sizeof(Time));
            fscanf(g,"%s\n",tc->string);
            tc->length = strlen(tc->string);
            op = create_op(oname,optype,x1,y1,x2,y2,tc);
            append_to_op_list(op_list,op);
        }
        else {
            dest = (Name *)malloc(sizeof(Name));
            fscanf(g,"%s\n",dest->string);
            dest->length = strlen(dest->string);
            dop = create_op(dest,optype,x1,y1,x2,y2,tc);
            ln = create_line(oname,optype,x1,y1,x2,y2,dop);
            append_line_to_op(op,ln);
        }
    }
    fclose(g);
    redraw_diagram();
}
```

153

```
dump_screen()
/*****************************************************************************
function:  performs a screen dump of the graphic editor display.
called by: notifier
calls:     none
*****************************************************************************/
{
    system(PRINT);
}




search()
/*****************************************************************************
function:  will interface with the CAPS database manager when the search
           button is selected
called by: notifier
calls:     none
*****************************************************************************/
{
}




compose()
/*****************************************************************************
function:  will perform the composition of a decomposed design when the
           compose button is selected
called by: notifier
calls:     none
*****************************************************************************/
{
}




decompose()
/*****************************************************************************
function:  will perform the composition of a decomposed design when the
           compose button is selected
called by: notifier
calls:     none
*****************************************************************************/
{
}
```

154

```
store_proc()
/********************************************************************************
function:  stores the data flow diargam into the design data base.
           Prior to storing the diagram it calls "create_PSDL" which
           transforms the picture into its equivalent PSDL statements.
called by: notifier
calls:     create_PSDL()
           store_diagram()
********************************************************************************/
{
    create_PSDL();
    store_diagram();
    graph_saved = 1;
}
```

```
static Notify_value
process_canvas_events(canvas,event)
    Canvas canvas;
    Event  *event;
/*******************************************************************************
function:  draws the graphical objects.
called by: notifier
calls:      pick_line()
            delete_line()
            pick_operator()
            delete_op()
            rubber_band()
            process_object()
            redraw_diagram()
*******************************************************************************/
{
    int id = event_id(event);
    static int x1, y1, x2, y2;
    static int new_posit = 1;
    static int left_button = 0;
    static int middle_button = 0;
    Operator *op;
    Line *ln;

    if (event_is_button(event)) {                       /* check for button events */
        if (event_is_down(event)) { /* store location where button goes down */
            x1 = event_x(event);                /* position of button down event
            y1 = event_y(event);
            switch(id) {
                case MS_LEFT:                               /* create new object */
                    new_posit =1;
                    x2 = x1;
                    y2 = y1;
                    left_button=1;
                    break;
                case MS_MIDDLE:                     /* pick object for moving */
                    break;
                case MS_RIGHT:                      /* pick object for deletion */
                    if ((ln = pick_line(op_list,x1,y1)) != NULL) {
                        op = NULL;
                        delete_line(op_list,op,ln);
                    }
                    else
                        if ((op = pick_operator(op_list,x1,y1)) != NULL)
                            delete_op(op_list,op);
                    redraw_diagram();
                    break;
            }
        }
        else if (event_is_up(event)) {
            switch (id) {
```

```
            case MS_LEFT:
                rubber_band(x1,y1,x2,y2);
                x2 = event_x(event);
                y2 = event_y(event);
                process_object(x1,y1,x2,y2);
                redraw_diagram();
                left_button=0;
                break;
            case MS_MIDDLE:                                  /* stubbed */
                if (!middle_button)
                    break;
            case MS_RIGHT:                                   /* stubbed */
                break;
        }
    }
}
else
    if (id == LOC_DRAG) {
        if (left_button) {
            if (!new_posit) {
                /* rubber band operator's boundary while being drawn */
                rubber_band(x1,y1,x2,y2);
                x2 = event_x(event);
                y2 = event_y(event);
                rubber_band(x1,y1,x2,y2);
            }
            else
                new_posit = 0;
        }
    }
return;
}
```

```c
process_object(x1,y1,x2,y2)
    int x1,y1,x2,y2;
/********************************************************************************
function:    processes operators, data flows, self loops, inputs and outputs
called by:   process_canvas_events()
calls:       pick_operator()
             process_operator()
             process_line()
             display_error_msg()
********************************************************************************/
{
    Operator *op, *sop, *dop;

    switch (edit_mode) {
        case OPERATOR:
            if (name_checked && time_checked) {
        /* draw object if it is not positioned on top of an existing object */
                if (((pick_operator(op_list,x1,y1))==NULL) &&
                    ((pick_operator(op_list,x2,y2))==NULL)) {
                        process_operator(OPERATOR,x1,y1,x2,y2);
                }
            }
            else {
                display_error_msg(4);
            }
            break;
        case DATA_FLOW:
            if (name_checked) {
                /* check if the line starts and terminates on an operator */
                if (((sop=pick_operator(op_list,x1,y1))!=NULL) &&
                    ((dop=pick_operator(op_list,x2,y2))!=NULL) && (sop != dop)) {
                    process_line(DATA_FLOW,x1,y1,x2,y2,sop,dop);
                }
            }
            else {
                display_error_msg(5);
            }
            break;
        case SELF_LOOP:
            if (name_checked) {
                /* draw the loop if it starts on an object and is not */
                /* intersecting an existing object */
                if (((sop=pick_operator(op_list,x1,y1))!=NULL) &&
                    ((dop=pick_operator(op_list,x2,y2))==NULL)) {
                    process_line(SELF_LOOP,x1,y1,x2,y2,sop,sop);
                }
            }
            else {
                display_error_msg(5);
            }
            break;
```

158

```c
case INPUT:
    if (name_checked) {
        /* check if line ends on an operator */
        if (((sop=pick_operator(op_list,x1,y1))==NULL) &&
            ((dop=pick_operator(op_list,x2,y2))!=NULL)) {
            process_line(INPUT,x1,y1,x2,y2,sop,dop);
        }
    }
    else {
        display_error_msg(5);
    }
    break;
case OUTPUT:
    if (name_checked) {
        dop = NULL;
        /* check if line is valid */
        if (((sop=pick_operator(op_list,x1,y1))!=NULL) &&
            ((dop=pick_operator(op_list,x2,y2))==NULL)) {
            process_line(OUTPUT,x1,y1,x2,y2,sop,dop);
        }
    }
    else {
        display_error_msg(5);
    }
    break;
default:
    break;
}
}
```

```
draw_object(otype,x1,y1,x2,y2)
    int   otype,x1,y1,x2,y2;
/*******************************************************************/
function:    draws object in the drawing space
called by:   process_operator()
             process_line()
             redraw_diagram()
calls:       none
/*******************************************************************/
{
    float i,  nid,ymid,xcent,ycent;
    int xnew,ynew,xold,yold;

    switch(otype) {
        case OPERATOR:
            xmid = (x2-x1)/2.0;
            ymid = (y2-y1)/2.0;              /* objects center point on the screen */
            xcent = x1 + xmid;
            ycent = y1 + ymid; /* find position to start drawing the object */
            xold = x2;
            yold = ycent;
            /* loop to draw the object */
            for(i = 0.0; i <= 2 * PI; i = i + PI / 12) {
                xnew = xcent + (xmid * cos(i));
                ynew = ycent + (ymid * sin(i));
                pw_vector(drawing_pw, xold, yold, xnew, ynew, PIX_SRC, 1);
                xold = xnew;
                yold = ynew;
            }
            break;
        case DATA_FLOW:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_SRC, 1);
            break;
        case SELF_LOOP:
            pw_vector(drawing_pw, x2, y1, x2, y2, PIX_SRC, 1);
            pw_vector(drawing_pw, x2, y2, x1, y2, PIX_SRC, 1);
            pw_vector(drawing_pw, x1, y2, x1, y1, PIX_SRC, 1);
            break;
        case INPUT:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_SRC, 1);
            break;
        case OUTPUT:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_SRC, 1);
            break;
        default :
            break;
    }
}
```

```
rubber_band(x1,y1,x2,y2)
    int  x1,y1,x2,y2;
/**********************************************************************
function:  expands/shrinks selected drawing object
called by: process_canvas_events()
calls:     none
**********************************************************************/
{
    switch(edit_mode) {
        case OPERATOR:
            pw_vector(drawing_pw, x1, y1, x2, y1, PIX_NOT(PIX_DST), 1);
            pw_vector(drawing_pw, x2, y1, x2, y2, PIX_NOT(PIX_DST), 1);
            pw_vector(drawing_pw, x2, y2, x1, y2, PIX_NOT(PIX_DST), 1);
            pw_vector(drawing_pw, x1, y2, x1, y1, PIX_NOT(PIX_DST), 1);
            break;
        case DATA_FLOW:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_NOT(PIX_DST), 1);
            break;
        case SELF_LOOP:
            pw_vector(drawing_pw, x2, y1, x2, y2, PIX_NOT(PIX_DST), 1);
            pw_vector(drawing_pw, x2, y2, x1, y2, PIX_NOT(PIX_DST), 1);
            pw_vector(drawing_pw, x1, y2, x1, y1, PIX_NOT(PIX_DST), 1);
            break;
        case INPUT:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_NOT(PIX_DST), 1);
            break;
        case OUTPUT:
            pw_vector(drawing_pw, x1, y1, x2, y2, PIX_NOT(PIX_DST), 1);
            break;
        default:
            break;

    }

}
```

161

```
process_operator(otype,x1,y1,x2,y2)
    int   otype,x1,y1,x2,y2;
/***********************************************************************
function:  checks the name and time constraint, stores into data structure
called by: process_object()
           draw_object()
calls:     draw_object()
           get_time_const()
           get_name()
           display_tc()
           create_op()
           append_to_op_list()
***********************************************************************/
{
    Name *obj_name;
    Time *tc;
    Operator *op;

    draw_object(otype,x1,y1,x2,y2);
    tc = get_time_const();
    obj_name = get_name();
    display_name(obj_name,otype,x1,y1,x2,y2);
    display_tc(tc,x1,y1,x2,y2);
    op = create_op(obj_name,otype,x1,y1,x2,y2,tc);
    append_to_op_list(op_list,op);
    name_checked = 0;
    time_checked = 0;
}
```

§

```
process_line(otype,x1,y1,x2,y2,sop,dop)
    int otype,x1,y1,x2,y2;
    Operator *sop,*dop;
/*********************************************************************
function:  gets attributes and draws a line
called by: process_object()
           draw_object()
calls:     draw_object()
           draw_arrowhead()
           get_time_const()
           external()
           create_op()
           append_to_op_list()
           get_name()
           display_name()
           create_line()
           append_to_op()
*********************************************************************/
{
    Name *obj_name,*op_name;
    Line *ln;

    draw_object(otype,x1,y1,x2,y2);
    if (otype == SELF_LOOP)
        draw_arrowhead(x2,y2,x2,y1);
    else
        draw_arrowhead(x1,y1,x2,y2);
    if (otype == INPUT) {
        tc = get_time_const();
        op_name = external();
        sop = create_op(op_name,EXTERNAL,x1,y1,x2,y2,tc);
        append_to_op_list(op_list,sop);
    }
    obj_name = get_name();
    display_name(obj_name,otype,x1,y1,x2,y2);
    ln = create_line(obj_name,otype,x1,y1,x2,y2,dop);
    append_line_to_op(sop,ln);
    name_checked = 0;
}
```

```
Operator *create_op(name,op_type,x1,y1,x2,y2,tc)
    Name    *name;
    int     op_type;
    int     x1,y1,x2,y2;
    Time    *tc;
/******************************************************************************
function:   gets the storage required to store an operator or input by
            calling alloc_operator. It then fills in the operator with its name,
            coordinates, and time constraint.
called by:  load_proc()
            process_operator()
            process_line()
calls:      alloc_operator()
******************************************************************************/
{
    Operator *new_op;
    new_op = alloc_operator();
    new_op->name      = name;
    new_op->time_const = tc;
    new_op->head = NULL;
    new_op->tail = NULL;
    new_op->next = NULL;
    switch(op_type) {
        case OPERATOR:
                    new_op->optype = OPERATOR;
                    new_op->xstart = x1;
                    new_op->ystart = y1;
                    new_op->xstop  = x2;
                    new_op->ystop  = y2;
                    break;
        case EXTERNAL:
                    new_op->optype = EXTERNAL;
                    new_op->xstart = 0;
                    new_op->ystart = 0;
                    new_op->xstop  = 0;
                    new_op->ystop  = 0;
                    break;
        default:
                    break;
    }
    return(new_op);
}
```

```
Line *create_line(name,ln_type,x1,y1,x2,y2,dest_op)
    Name      *name;
    int       ln_type;
    int       x1,y1,x2,y2;
    Operator *dest_op;
/*******************************************************************************
function:  gets the storage required to store a data flow line or an
           output line by calling alloc_line. It then fills in the line struct
           with its name and coordinates.
called by: load_proc()
           process_line()
calls:     alloc_line()
           external()
*******************************************************************************/
{
    Line *new_ln;

    new_ln = alloc_line();
    new_ln->name   = name;
    new_ln->xstart = x1;
    new_ln->ystart = y1;
    new_ln->xstop  = x2;
    new_ln->ystop  = y2;
    new_ln->next   = NULL;
    switch(ln_type) {
        case INPUT:
                    new_ln->lntype = INPUT;
                    new_ln->dest   = dest_op->name;
                    break;
        case DATA_FLOW:
                    new_ln->lntype = DATA_FLOW;
                    new_ln->dest   = dest_op->name;
                    break;
        case OUTPUT:
                    new_ln->lntype = OUTPUT;
                    new_ln->dest   = external();
                    break;
        case SELF_LOOP:
                    new_ln->lntype = SELF_LOOP;
                    new_ln->dest   = dest_op->name;
                    break;
        default:
                    break;
    }
    return(new_ln);
}
```

```
delete_op(op_list,op)
    Operator_list *op_list;
    Operator *op;
/*******************************************************************************
function:   deletes operators from the drawing space and the internal data
            structure
called by:  process_canvas_events()
            delete_input_lines()
calls:      delete_input_lines()
*******************************************************************************/
{
    Operator *dptr,
             *otemp;
    Line     *lptr,
             *ltemp;
    Name     *n;

    /* find lines which terminate on this operator and delete them */
    n = op->name;
    delete_input_lines(op_list,n);
    otemp = op_list->head;                  /* put pointer at head of op list */
    if (op != otemp) {                      /* is the first op the one to delete? */
        while (otemp->next != op) {         /* if not, find the one to delete */
            otemp = otemp->next;
        }
        dptr = otemp->next;                 /* unlink the op to be deleted   */
        if (dptr->next != NULL) {
            otemp->next = dptr->next;
            dptr->next = NULL;
        }
        else
            otemp->next = NULL;
    }
    else {                                  /* the first one is the one to delete */
        dptr = op_list->head;                       /* unlink the first op */
        op_list->head = dptr->next;
    }
    if (dptr->head != NULL) {               /* does it have any assoc lines ? */
        ltemp = dptr->head;
        lptr = dptr->head;
        dptr->head = NULL;
        while(lptr->next != NULL) {
            lptr = lptr->next;
            ltemp = lptr;
        }
        lptr = NULL;
        ltemp = NULL;
    }
    dptr = NULL;
}
```

```c
delete_input_lines(op_list,n)
    Operator_list *op_list;
    Name *n;
/*********************************************************************
function:  deletes lines associated with an operator when deleting operators
called by: delete_op()
calls:     delete_op()
           delete_line()
*********************************************************************/
{
    Operator *optr;
    Line     *lptr,
             *ltemp;

    optr = op_list->head;
    while(optr != NULL) {                   /* search the entire list of operators */
        lptr = optr->head;
        while(lptr != NULL) {          /* check each line leaving each operator */
            if(!strcmp(n->string,lptr->dest->string)) {
                ltemp = lptr->next;
                /* found a line with the destination searched for */
                delete_line(op_list,optr,lptr);           /* so delete it */
    /*          if(optr->optype == EXTERNAL) {
                    delete_op(op_list,optr);
                )     */
                lptr = ltemp;
            }
            else {
                lptr = lptr->next;
            }
        }
        optr = optr->next;
    }
}
```

167

```
delete_line(op_list,op,ln)
    Operator_list *op_list;
    Operator *op;
    Line *ln;
/*******************************************************************************
function:   removes line from linked list data structure
called by: delete_input_lines()
calls:      none
*******************************************************************************/
{
    Operator *optr;
    Line      *lptr,
              *ltemp;
    int       ln_found = 0;

    if(op != NULL) {
        optr = op;             /* start the search for the line at its source op */
        lptr = optr->head;
        ltemp = lptr;
        while(lptr != ln) {
            ltemp = lptr;
            lptr = lptr->next;
        }
    }
    else {                                      /* source op is unknown - find the line */
        optr = op_list->head;
        while((optr != NULL) && (!ln_found)) {
            lptr = optr->head;
            ltemp = lptr;
            while((lptr != NULL) && (!ln_found)) {
                if (lptr == ln)
                    ln_found = 1;
                else {
                    ltemp = lptr;
                    lptr = lptr->next;
                }
            }
            if (!ln_found)
                optr = optr->next;
        }
    }
    /* unlink the line */
    if (ltemp == lptr) {                        /* delete first line on list */
        optr->head = ltemp->next;
        lptr->next = NULL;
        ltemp = NULL;
        if (optr->head == NULL) {               /* first line was the only line */
            optr->tail = NULL;
        }
    }
    else {
```

168

```
        if(lptr == optr->tail) {      /* delete last line from the list */
            ltemp->next = NULL;
            optr->tail = ltemp;
        }
        else {                         /* delete a middle line from the list */
            ltemp->next = lptr->next;
            lptr->next = NULL;
        }
        optr = NULL;
    }
}
```

```
Operator *pick_operator(op_list,xpick,ypick)
    Operator_list *op_list;
    int           xpick, ypick;
/****************************************************************************
function:  determines if a data flow line or output line starts on an
           operator. If the search for a source operator is successful, it
           returns a pointer to that operator.
called by: process_canvas_events()
           process_operator()
calls:     is_op_pick()
****************************************************************************/
{
    Operator *ptr;

    for (ptr = op_list->head; ptr != NULL; ptr = ptr->next) {
        if (ptr->optype == OPERATOR) {              /* skip the null operators */
            if (is_op_pick(ptr->xstart, ptr->ystart,       /* test for pick */
                ptr->xstop, ptr->ystop, xpick, ypick)) {
                return(ptr);
            }
        }
    }
    return(NULL);
}
```

```c
int is_op_pick(x1,y1,x2,y2,xp,yp)
    int    x1, y1, x2, y2, xp, yp;
/************************************************************************
function:  determines if a pick has occurred within the bounds of an
           operator.
called by: pick_operator()
calls:     none


*************************************************************************/
{
    if (( (xp > x1) && (xp < x2) && (yp > y1) && (yp < y2) )||
        ( (xp < x1) && (xp > x2) && (yp > y1) && (yp < y2) )||
        ( (xp < x1) && (xp > x2) && (yp < y1) && (yp > y2) )||
        ( (xp > x1) && (xp < x2) && (yp < y1) && (yp > y2) ))
        return(1);
    else
        return(0);
}
```

171

```
Line *pick_line(op_list,xpick,ypick)
/*************************************************************************
function:  determines if a line object was picked with the mouse
called by: peocess_canvas_events()
calls:     is_line_pick()
*************************************************************************/
Operator_list *op_list;
int            xpick, ypick;

{
    Operator *optr;
    Line      *lptr;

    /* search each operator's line list */
    for (optr = op_list->head; optr != NULL; optr = optr->next) {
        for (lptr = optr->head; lptr != NULL; lptr = lptr->next) {
            if (is_line_pick(lptr->xstart, lptr->ystart,  /* test for pick */
                lptr->xstop, lptr->ystop, xpick, ypick)) {
                return(lptr);
            }
        }
    }
    return(NULL);
}
```

172

```c
int is_line_pick(x1,y1,x2,y2,xp,yp)
    int   x1, y1, x2, y2, xp, yp;
/***********************************************************************
function:  determines if mouse is on a line
called by: pick_line()
calls:     none
***********************************************************************/
{
    if (((abs(x1-xp)+abs(y1-yp)) < PROXIMITY) ||
        ((abs(x2-xp)+abs(y2-yp)) < PROXIMITY))
        return(1);
    else
        return(0);
}
```

```
append_to_op_list(op_list,op)
    Operator_list *op_list;
    Operator *op;
/*******************************************************************************
function:  adds new operators to linked list of operators
called by: load_proc()
           process_operator()
           process_line()
calls:     none
*******************************************************************************/
{
    if (op_list->head == NULL) {
        op_list->head = op;                 /* attach first operator to list */
        op_list->tail = op;
    }
    else {
        op_list->tail->next = op;           /* attach operator to end of list */
        op_list->tail = op;
    }
}
```

```
append_line_to_op(op,ln)
    Operator *op;
    Line      *ln;
/***************************************************************************
function:  attaches data flows, states, and outputs to link list of operators
called by: load_proc()
           process_line()
calls:     none
***************************************************************************/
{
    if (op->head == NULL) {
        op->head = ln;                          /* attach first line to list */
        op->tail = ln;
    }
    else {
        op->tail->next = ln;                     /* attach to end of line list */
        op->tail = ln;
    }
}
```

```
Operator *alloc_operator()
/******************************************************************
function:   allocates dynamic storage for operators their inputs
called by: create_op()
calls:      none
******************************************************************/
{
    Operator *op;

    op = (Operator *)malloc(sizeof(Operator));
    return(op);
}
```

```
Line *alloc_line()
/**********************************************************************
function:  allocates dynamic storage for data flows and outputs of an operator
called by: create_line()
calls:     none
**********************************************************************/
{
    Line *ln;

    ln = (Line *)malloc(sizeof(Line));
    return(ln);
}
```

```
input_text(item, value, event)
    Panel_item    item;
    int           value;
    Event         *event;
/************************************************************************
function:   reads the names from the name panel
called by:  notifier when entering an object identifier name
calls:      is_valid_ada_id()
            display_error_msg()
************************************************************************/
{
    tmp_buf = malloc(80);
                                                /* initialize the storage */
    tmp_buf[0] = ' ';
    strcpy(tmp_buf, (char *)panel_get_value(object_name));
    /* check to see if the name is an ada identifier */
    if (is_valid_ada_id(tmp_buf)) {
        display_error_msg(1);
        name_checked = 1;
    }
    else
        display_error_msg(2);
}
```

```c
display_error_msg(msg_id)
    int    msg_id;
/***********************************************************************
function:  displays warnings and error messages in the message panel
called by: process_object()
           input_text()
           input_time()
           quit_proc()
calls:     none
***********************************************************************/
{
    char   *msg;

    msg = malloc(61);
    switch(msg_id) {
        case 1:
            msg = " MESSAGE PANEL:                                      ";
            break;
        case 2:
            msg = " MESSAGE PANEL: SYNTAX ERROR in ADA identifier       ";
            break;
        case 3:
            msg = " MESSAGE PANEL: SYNTAX ERROR in Maximum Execution Time   ";
            break;
        case 4:
            msg = " MESSAGE PANEL: ERROR - Either NAME or TIME not read   ";
            break;
        case 5:
            msg = " MESSAGE PANEL: ERROR - NAME not read                ";
            break;
        case 6:
            msg = " MESSAGE PANEL: WARNING - The graph has not been stored!   ";
            break;
        default:
            break;
    }
    window_set(message_panel,0);
    panel_set(message, PANEL_LABEL_STRING, msg, 0);
}
```

```
Name *get_name()
/*****************************************************************************
function:   creates a dynamic name structure
called by:  process_operator()
            process_line()
calls:      none
*****************************************************************************/
{
    Name *n;

    n = (Name *)malloc(sizeof(Name));
    strcpy(n->string,tmp_buf);
    n->length = strlen(n->string);
    return(n);
}
```

```
display_name(n,otype,x1,y1,x2,y2)
    Name    *n;
    int     otype,x1,y1,x2,y2;
/********************************************************************
function:  displays the name of the object which the user has drawn.
           Operator names are centered within the operator, data flow line
           names start above the center of the line, input names start above
           the initial point of the line, and output names start at the end
           of the output line.
called by: process_operator()
           process_line()
           redraw_diagram()
calls:     none
********************************************************************/
{
    float   xcent,ycent;

    xcent = x1 + ((x2 - x1) / 2.0);
    ycent = y1 + ((y2 - y1) / 2.0);
    switch(otype) {
        case OPERATOR:
                    xcent = x1 + ((x2 - x1) / 2.0);
                    ycent = y1 + ((y2 - y1) / 2.0);
                    pw_text(canvas_pixwin(drawing_canvas),
                            (int)xcent-((n->length)/2)*8,
                            (int)ycent+5,PIX_SRC,NULL,n->string);
                    break;
        case INPUT:
                    pw_text(canvas_pixwin(drawing_canvas),x1,y1,PIX_SRC,
                            NULL,n->string);
                    break;
        case OUTPUT:
                    pw_text(canvas_pixwin(drawing_canvas),x2,y2,PIX_SRC,
                            NULL,n->string);
                    break;
        case DATA_FLOW:
                    xcent = (x2 - x1) / 2.0;
                    ycent = (y2 - y1) / 2.0;
                    pw_text(canvas_pixwin(drawing_canvas),x1+(int)xcent,
                            y1+(int)ycent,PIX_SRC,NULL,n->string);
                    break;
        case SELF_LOOP:
                    xcent = x1 + ((x2 - x1) / 2.0);
                    pw_text(canvas_pixwin(drawing_canvas),
                            (int)xcent-((n->length)/2)*8,
                            (int)y2-7,PIX_SRC,NULL,n->string);
                    break;
        default:
                    break;
    }
}
```

```c
int is_valid_ada_id(tmp_buf)
    char   tmp_buf[00];
/*********************************************************************
function:  checks to see if a name is a legal ada identifier
called by: input_text()
calls:     none
*********************************************************************/
{
    int    space_found = 0;
    int    i = 1;

    if (isalpha(tmp_buf[0])) {
        while (i <= strlen((char *)tmp_buf) - 1) {
            if(!isgraph(tmp_buf[i])) {
                space_found = 1;
                i = i + 1;
            }
            else {
                if((((isalnum(tmp_buf[i]))||(tmp_buf[i]=='_'))&& !(space_found))
                    i = i + 1;
                else
                    return(0);
            }
        }
        return(1);
    }
    else
        return(0);
}
```

182

```
input_time(item, value, event)
    Panel_item    item;
    int           value;
    Event         *event;
/***********************************************************************
function:  gets and checks time constraints
called by: notifies
calls:     display_error_msg()
           is_valid_time_const()
***********************************************************************/
{
    tmp_buf1 = malloc(12);
    tmp_buf1[0] = ' ';
    strcpy(tmp_buf1, (char *)panel_get_value(time_constraint));
    if (is_valid_time_const(tmp_buf1)) {
        display_error_msg(1);
        time_checked = 1;
    }
    else
        display_error_msg(3);
}
```

```c
int is_valid_time_const(tmp_buf1)
    char   tmp_buf1[12];
/********************************************************************************
function:  checks syntax for time constraint
called by: input_time()
calls:     none
********************************************************************************/
{
    int    nondigit_found = 0;
    int    letter_prev_found = 0;
    int    mfound = 0;
    int    ufound = 0;
    int    done = 0;
    int    i = 1;

    if (isdigit(tmp_buf1[0])) {
        while (i <= strlen((char *)tmp_buf1) - 1) {
            if (!isalnum(tmp_buf1[i]))
                return(0);
            else {
                if (isdigit(tmp_buf1[i])&&!nondigit_found)
                    i = i + 1;
                else {
                    if (isdigit(tmp_buf1[i]))
                        return(0);
                    else {
                        nondigit_found = 1;
                        switch(tmp_buf1[i]) {
                        case 'u': if (!letter_prev_found) {
                                    ufound = 1;
                                    letter_prev_found = 1;
                                    i = i + 1;
                                }
                                else
                                    return(0);
                                break;
                        case 's': if (!letter_prev_found) {
                                    letter_prev_found = 1;
                                    i = i + 1;
                                }
                                else {
                                    if ((ufound||mfound)&&!done) {
                                        done = 1;
                                        i = i + 1;
                                    }
                                    else
                                        return(0);
                                }
                                break;
                        case 'm': if (!letter_prev_found) {
                                    mfound = 1;
```

```
                                        letter_prev_found = 1;
                                        i = i + 1;
                                }
                                else
                                        return(0);
                                break;
                          default : return(0);
                                break;
                        }
                }
            }
        }
    }
    if ((ufound&&!done)||(!nondigit_found))
        return(0);
}
else
    return(0);
}
```

```c
Time *get_time_const()
/*********************************************************************
function:  creates initial time structure for objects
called by: process_operator()
           process_line()
calls:     none
*********************************************************************/
{
    Time *tc;

    /* get storage for the time constraint */
    tc = (Time *)malloc(sizeof(Time));
    switch(edit_mode) {
        case OPERATOR:
            strcpy(tc->string,tmp_bufl); /* assign input string */
            tc->length = strlen(tc->string); /* find length of the string */
            break;
        case INPUT:
            strcpy(tc->string,"0s");
            tc->length = 2;
            break;
        default:
            break;
    }
    return(tc);
}
```

```
display_tc(tc,x1,y1,x2,y2)
    Time *tc;
    int x1,y1,x2,y2;
/***************************************************************************
function:   displays time constraint in the drawing space
called by:  process_operator()
            redraw_diagram()
calls:      none
***************************************************************************/
(
    float   xcent;

    xcent = x1 + ((x2 - x1) / 2.0);
    pw_text(canvas_pixwin(drawing_canvas), (int)xcent-((tc->length)/2)*8,
        (int)y1-5,PIX_SRC,NULL,tc->string);
)
```

```
Name *external()
/********************************************************************
function:  returns the name "EXTERNAL" whenever it is called. It is
           used to provide the names for the source of input lines and
           destination of output lines.
called by: create_line()
           process_line()
calls:     none
********************************************************************/
{
    Name *n;

    n = (Name *)malloc(sizeof(Name));               /* alloc storage */
    strcpy(n->string,"EXTERNAL");                     /* assign name */
    n->length = 8;                                    /* assign name */
    return(n);
}
```

```
create_PSDL()
/******************************************************************************
function:   creates the PSDL statements represented by the user's data
            flow diagram. A PSDL statement of the form
            output_line_name.source_name[:time_constraint]->destination_name
            will be constructed from the information contained in the operator
            list.
called by: store_proc()
calls:     none
******************************************************************************/
{
    Operator  *op_ptr;
    Line      *output_ptr;
    char      *psdl;
    char      *period = ".";
    char      *colon  = ":";
    char      *arrow  = "->";

    f = fopen("/n/suns2/work/caps/prototypes/graph.links","w");
    psdl = malloc(270);
    psdl[0] = ' ';
    op_ptr = op_list->head;                /* point at head of the operator list */
    while(op_ptr != NULL) {
        output_ptr = op_ptr->head;         /* point line ptr at head of line list*/
        while(output_ptr != NULL) {
            /* assemble the psdl statement by concatenating the parts of the
               PSDL statements together */
            psdl = strcat(psdl,output_ptr->name->string);
            psdl = strcat(psdl,period);
            psdl = strcat(psdl,op_ptr->name->string);
            psdl = strcat(psdl,colon);
            if ((op_ptr->optype == OPERATOR) &&
                (op_ptr->time_const->string != "0s"))
                psdl = strcat(psdl,op_ptr->time_const->string);
            psdl = strcat(psdl,arrow);
            psdl = strcat(psdl,output_ptr->dest->string);
            fprintf(f,"%s\n",psdl);                     /* store link stmt in file */
            psdl[0] = ' ';                              /* reinitialize */
            output_ptr = output_ptr->next;
        }
        op_ptr = op_ptr->next;
    }
    fclose(f);
}
```

```
store_diagram()
/******************************************************************************
function:  writes the current prototype design to a file
called by: store_proc()
calls:     none
******************************************************************************/
{
    Operator *op_ptr;
    Line     *ln_ptr;

    g = fopen("/n/suns2/work/caps/prototypes/graph.pic","w");
    op_ptr = op_list->head;
    while(op_ptr != NULL) {
        fprintf(g,"%d\n",op_ptr->optype);
        fprintf(g,"%d\n",op_ptr->xstart);
        fprintf(g,"%d\n",op_ptr->ystart);
        fprintf(g,"%d\n",op_ptr->xstop);
        fprintf(g,"%d\n",op_ptr->ystop);
        fprintf(g,"%s\n",op_ptr->name->string);
        fprintf(g,"%s\n",op_ptr->time_const->string);
        ln_ptr = op_ptr->head;
        while(ln_ptr != NULL) {
            fprintf(g,"%d\n",ln_ptr->lntype);
            fprintf(g,"%d\n",ln_ptr->xstart);
            fprintf(g,"%d\n",ln_ptr->ystart);
            fprintf(g,"%d\n",ln_ptr->xstop);
            fprintf(g,"%d\n",ln_ptr->ystop);
            fprintf(g,"%s\n",ln_ptr->name->string);
            fprintf(g,"%s\n",ln_ptr->dest->string);
            ln_ptr = ln_ptr->next;
        }
        op_ptr = op_ptr->next;
    }
    fclose(g);
}
```

```
redraw_diagram()
/**************************************************************************
function:    redraws the diagram in the drawing space
called by:   load_proc()
             process_canvas_events()
calls:       draw_object()
             display_name()
             display_tc()
             draw_arrowhead()
**************************************************************************/
{
    Operator *op_ptr;
    Line     *ln_ptr;

    pw_writebackground(drawing_pw,0,0,
                       window_get(drawing_canvas,CANVAS_WIDTH),
                       window_get(drawing_canvas,CANVAS_HEIGHT),
                       PIX_SRC);
    op_ptr = op_list->head;
    while(op_ptr != NULL) {
        if(op_ptr->optype == OPERATOR) {
            draw_object(op_ptr->optype,op_ptr->xstart,op_ptr->ystart,
                        op_ptr->xstop,op_ptr->ystop);
            display_name(op_ptr->name,OPERATOR,op_ptr->xstart,
                         op_ptr->ystart,op_ptr->xstop,op_ptr->ystop);
            display_tc(op_ptr->time_const,op_ptr->xstart,op_ptr->ystart,
                       op_ptr->xstop,op_ptr->ystop);
        }
        ln_ptr = op_ptr->head;
        while(ln_ptr != NULL) {
            draw_object(ln_ptr->lntype,ln_ptr->xstart,ln_ptr->ystart,
                        ln_ptr->xstop,ln_ptr->ystop);
            if (ln_ptr->lntype == SELF_LOOP)
                draw_arrowhead(ln_ptr->xstop,ln_ptr->ystop,
                               ln_ptr->xstop,ln_ptr->ystart);
            else
                draw_arrowhead(ln_ptr->xstart,ln_ptr->ystart,
                               ln_ptr->xstop,ln_ptr->ystop);
            display_name(ln_ptr->name,ln_ptr->lntype,ln_ptr->xstart,
                         ln_ptr->ystart,ln_ptr->xstop,ln_ptr->ystop);
            ln_ptr = ln_ptr->next;
        }
        op_ptr = op_ptr->next;
    }
}
```

191

```
draw_arrowhead(x1, y1, x2, y2)
    int x1, y1, x2, y2;
/***************************************************************************
function:  draws an arrow head at the end of a line at the appropriate angle
called by: process_line()
           redraw_arrowhead()
calls:     none
***************************************************************************/
{
    int    x1_trans, y1_trans, x2_trans, y2_trans;
    int    xpt1, ypt1, xpt2, ypt2,
           xpt1_trans, ypt1_trans, xpt2_trans, ypt2_trans;
    double length, theta;

    /* translate the line to the origin */
    x1_trans = x1 - x2;
    y1_trans = y1 - y2;
    /* find the length of the line */
    length = sqrt(pow((double)x1_trans,2.0) + pow((double)y1_trans,2.0));
    /* find the angle between the line and the x axis */
    theta = acos ((double)x1_trans/length);
    /* calculate the coords of the points of the arrowhead */
    xpt1 = ARROW_LENGTH * cos(theta + PI / 6.0);
    ypt1 = ARROW_LENGTH * sin(theta + PI / 6.0);
    xpt2 = ARROW_LENGTH * cos(theta - PI / 6.0);
    ypt2 = ARROW_LENGTH * sin(theta - PI / 6.0);
    /* reflect y coords across x axis if y1_trans is negative */
    if (y1_trans < 0) {
        ypt1 = -ypt1;
        ypt2 = -ypt2;
    }
    /* translate the coords of the arrowhead out to the posit of the line */
    xpt1_trans = xpt1 + x2;
    ypt1_trans = ypt1 + y2;
    xpt2_trans = xpt2 + x2;
    ypt2_trans = ypt2 + y2;
    /* draw the point of the arrow */
    pw_vector(drawing_pw, xpt1_trans, ypt1_trans, x2, y2, PIX_SRC, 1);
    pw_vector(drawing_pw, xpt2_trans, ypt2_trans, x2, y2, PIX_SRC, 1);
    pw_vector(drawing_pw, xpt1_trans, ypt1_trans, xpt2_trans, ypt2_trans,
            PIX_SRC, 1);
}
```

APPENDIX E LINK STATEMENT ANALYZER

```
(*------------------------------------------------------------------
Program: nodes.p
Author:  Hank Raum
Last Modified: 9 December 89 by Laura J. White
--------------------------------------------------------------------*)

program CreateNodes (input,output);

const                   (* Global Constants *)
   period = '.';
   colon = ':';
   arrow = '-';
   blank = ' ';
   EXTERNAL = 'EXTERNAL

type
   string80 = packed array [0..79] of char;
   DataPtr = ^DataType;
   DataType = record                      (* Node for Linked List *)
      Name: string80;                     (*      of Nodes *)
      Link: DataPtr;
      end;  (* DataType *)
   OperPtr = ^Operator;
   Operator = record              (* Node of Linked List of Operators *)
      OpName: string80;               (* Operator Name *)
      InputList: DataPtr;             (* Head Pointer to Input List *)
      InListTail: DataPtr;            (* Tail Pointer to Input List *)
      OutputList: DataPtr;            (* Head Pointer to Output List *)
      OutListTail: DataPtr;           (* Tail Pointer to Output List *)
      StateList: DataPtr;             (* Head Pointer to State List *)
      StateListTail: DataPtr;         (* Tail Pointer to State List *)
      MET: string80;                  (* Maximum Execution Time *)
      Link: OperPtr;
      end;  (* Operator *)

var
   OpHead: OperPtr;              (* Head of Operator List *)
   OpTail: OperPtr;              (* Tail of Operator List *)
   DataHead: DataPtr;            (* Head of Data List *)
   DataTail: DataPtr;           (* Tail of Data List *)


(*------------------------------------------------------------*)
```

193

```
procedure ReadToken(delimeter:char;
                    var token:string80);

(* Reads PSDL Link statements from standard input, one token at  *)
(* a time.  Delimeters are: period, colon, arrow and End of Line *)

var
   ndx:integer;
   ch: char;

begin
   ndx := 0;                   (* initialize *)
   read(ch);
   while (ch <> delimeter) and (not eoln) do
      begin
      token[ndx] := ch;        (* Gets token character by character *)
      read(ch);                (* until delimeter or eoln *)
      ndx := ndx + 1;
      end;    (* while*)
   if eoln then token[ndx] := ch;    (* Gets last character before *)
   if delimeter = arrow then              (* end of line *)
      begin
      read(ch); read(ch);              (* remove rest of arrow *)
      end;    (* if *)
   if eoln then readln;                 (* resets line *)
end;    (* ReadToken *)

(*-----------------------------------------------------------*)
```

194

```
procedure ReadOperMet(var Operl, Met: string80);

   (* Reads PSDL Link statements from standard input, one token at  *)
   (* a time.  Determines Operatorl and Maximum Execution Time   *)

   var
      ndx:integer;
      ch: char;

   begin
      ndx := 0;                    (* initialize *)
      read(ch);
      while (ch <> colon) and (ch <> arrow) do
         begin
         Operl[ndx] := ch;         (* Gets token character by character *)
         read(ch);                 (* until delimeter or  oln *)
         ndx := ndx + 1;
         end;    (* while*)
      if ch = colon then              (* end of line *)
         begin
         ndx := 0;
         read(ch);
         while ch <> arrow do
            begin
            Met[ndx] := ch;           (* Gets token by character *)
            read(ch);                  (* until delimeter or eoln *)
            ndx := ndx + 1;
            end;     (* while*)
         end;     (* if *)
      read(ch); read(ch);            (* remove rest of arrow *)
   end;    (* ReadToken *)

(*------------------------------------------------------------*)
```

195

```
function OpSearch (Head: OperPtr;
                  Target: string80): OperPtr;

   (* Searches Operator List for Target string, returns pointer *)
   (* to target if found, otherwise NIL *)

begin
   if Head = nil then
      OpSearch := nil                (* empty list *)
   else if Head^.OpName = Target then
      OpSearch := Head               (* target found *)
   else
      OpSearch := OpSearch(Head^.Link, Target);
end;
```

(*--*)

196

```
procedure OpAdd (var Head: OperPtr;
                 var Tail: OperPtr;
                   Target: string80);

    (* Adds new Operator to end of linked list *)

var
    p: OperPtr;                      (* temp pointer *)

begin
    if Head = nil then               (* List is empty *)
        begin
        new(p);                      (* Create new head node *)
        Head := p;
        Tail := p;
        p^.OpName := Target;         (* Initialize new list *)
        p^.InputList := nil;
        p^.InListTail := nil;
        p^.OutputList := nil;
        p^.OutListTail := nil;
        p^.Link := nil;
        end  (* if *)
    else                             (* List not empty *)
        begin
        new(p);                      (* Add new node after tail *)
        Tail^.Link := p;
        Tail := Tail^.Link;
        p^.OpName := Target;         (* Initialize new lists *)
        p^.InputList := nil;
        p^.InListTail := nil;
        p^.OutputList := nil;
        p^.OutListTail := nil;
        p^.StateList := nil;
        p^.StateListTail := nil;
        p^.Link := nil;
        end (* else *)
    end;    (* OpAdd *)

(*---------------------------------------------------------------*)
```

```
function Search (Head: DataPtr;
                 Target: string80): DataPtr;

    (* Searches Data Listfor Target string, returns pointer  *)
    (* to target if found, otherwise NIL *)

begin
   if Head = nil then
      Search := nil              (* empty list *)
   else if Head^.Name = Target then
    Search := Head               (* target found *)
   else
    Search := Search(Head^.Link, Target);
end;   (* Search *)

(*-------------------------------------------------------------*)
```

198

```
procedure Add (var Head: DataPtr;
               var Tail: DataPtr;
                   Target: string80);

    (* Adds new Data to end of linked lists *)

var
   p: DataPtr;                    (* Temp pointer *)

begin
   if Head = nil then             (* List is empty *)
      begin
      new(p);                     (* Create new node *)
      Head := p;
      Tail := p;
      p^.Name := Target;          (* Initialize new lists *)
      p^.Link := nil;
     end    (* if *)
   else                           (* List not empty *)
      begin
      new(p);                     (* Add new node after tail *)
      Tail^.Link := p;
      Tail := Tail^.Link;
      p^.Name := Target;          (* Initialize new lists *)
      p^.Link := nil;
      end (* else *)
   end;    (* OpAdd *)

(*------------------------------------------------------------*)
```

```
procedure LoadDataStructure(var OpHead, OpTail: OperPtr;
                            var DataHead, DataTail: DataPtr);

    (* Loads tokens into Data Structures *)

var
    Current: OperPtr;              (* Temp pointer *)
    Data, Met:  string80;          (* FSDL Tokens *)
    Oper1, Oper2:  string80;

begin
    Data := blank;                 (* Initialize Strings *)
    Oper1 := blank;
    Met := blank;
    Oper2 := blank;

    while not eof do
        begin                      (* Get tokens *)
        ReadToken(period,Data);
        ReadOperMet(Oper1,Met);
        ReadToken(' ',Oper2);
        if Oper1 <> EXTERNAL then   (* Keyword EXTERNAL is not *)
            begin                   (* an Operator *)
                (*********** segv on next statement ***********)
            Current := OpSearch(OpHead,Oper1);
            if Current = nil then
                begin               (* Add Operator 1 *)
                OpAdd(OpHead,OpTail,Oper1);
                Current := OpSearch(OpHead,Oper1);
                end;   (* if *)
            Current^.MET := Met;    (* Enter Maximun Execution Time *)
                            (* Add Data to Operators Output List *)
            if Oper1 = Oper2 then
                begin
                if Search(Current^.StateList,Data) = nil then
                    Add(Current^.StateList,Current^.StateListTail,
                        Data);
                end
            else
                if Search(Current^.OutputList,Data) = nil then
                    Add(Current^.OutputList,Current^.OutListTail,
                        Data);
            end;    (* if *)
        if Oper2 <> EXTERNAL then   (* Keyword EXTERNAL is not *)
            begin                   (* an Operator *)
            Current := OpSearch(OpHead,Oper2);
            if Current = nil then
                begin               (* Add Operator 2 *)
                OpAdd(OpHead,OpTail,Oper2);
                Current := OpSearch(OpHead,Oper2);
                end;    (* if *)
```

```
                                        (* Add Data to Operators Input List *)
        if Oper1 = Oper2 then
           begin
           if Search(Current^.StateList,Data) = nil then
              Add(Current^.StateList,Current^.StateListTail,
                 Data);
           end
        else
           if Search(Current^.InputList,Data) = nil then
              Add(Current^.InputList,Current^.InListTail,
                 Data);
        end;    (* if *)
           (* Enter new internal Data Streams in Data List *)
        if ((Oper1 <> EXTERNAL) and (Oper2 <> EXTERNAL)) and
           (Oper1 <> Oper2) then
           if Search(DataHead,Data) = nil then
              Add(DataHead,DataTail,Data);

     Data := blank;                   (* Reset Strings *)
     Oper1 := blank;
     Met := blank;
     Oper2 := blank;
     end;    (* while *)
  end;    (* LoadDataStructure *)


(*----------------------------------------------------------*)
```

```
procedure WriteString(var File:text; Str: string80);

var
   ndx: integer;

begin
   ndx := 0;
   while Str[ndx] <> ' ' do
      begin
      write(File,Str[ndx]);
      ndx := ndx + 1;
      end;    (* while *)
   end;    (* WriteString *)

(*--------------------------------------------------------------*)
```

```
procedure MakePSDL(Head: OperPtr);

   (* Generates partial PSDL Specification for each new Operator *)
   (* in the Graphical decomposition *)

type
   string42 = packed array [0..41] of char;

var
   Current: OperPtr;              (* Temp pointers *)
   InTemp:  DataPtr;
   OutTemp: DataPtr;
   StateTemp: DataPtr;
   OutFile: text;
   NodeName: string42;           (* Unix file name *)

begin
   Current := Head;
   NodeName := '/n/suns2/work/caps/prototypes/NewNode.01';

   while Current <> nil do
      begin
      rewrite(OutFile,NodeName);   (* Create new file *)
           (* output PSDL *)
      write(OutFile,'OPERATOR ');
      WriteString(OutFile,Current^.OpName);
      writeln(OutFile);
      writeln(OutFile);
      writeln(OutFile,'SPECIFICATION');
      writeln(OutFile);
      InTemp := Current^.InputList;
      if InTemp <> nil then          (* Generate Input list *)
         begin
         write(OutFile,'INPUT ');
         WriteString(OutFile,InTemp^.Name);
         writeln(OutFile);
         InTemp := InTemp^.Link;
         while InTemp <> nil do
            begin
            write(OutFile,'        ');
            WriteString(OutFile,InTemp^.Name);
            writeln(OutFile);
            InTemp := InTemp^.Link;
            end;    (* while *)
         writeln(OutFile);
         end;    (* if *)
      OutTemp := Current^.OutputList;
      if OutTemp <> nil then
         begin                         (* Generate Output list *)
         write(OutFile,'OUTPUT ');
         WriteString(OutFile,OutTemp^.Name);
```

203

```
                        writeln(OutFile);
                        OutTemp := OutTemp".Link;
                        while OutTemp <> nil do
                            begin
                            write(OutFile,'            ');
                            WriteString(OutFile,OutTemp".Name);
                            writeln(OutFile);
                            OutTemp := OutTemp".Link;
                            end;    (* while *)
                        writeln(OutFile);
                        end;    (* if *)
                    StateTemp := Current".StateList;
                    if StateTemp <> nil then
                        begin                        (* Generate State list *)
                        write(OutFile,'STATE ');
                        WriteString(OutFile,StateTemp".Name);
                        writeln(OutFile);
                        StateTemp := StateTemp".Link;
                        while StateTemp <> nil do
                            begin
                            write(OutFile,'            ');
                            WriteString(OutFile,StateTemp".Name);
                            writeln(OutFile);
                            StateTemp := StateTemp".Link;
                            end;    (* while *)
                        writeln(OutFile);
                        end;    (* if *)
                    write(OutFile,'MAXIMUM EXECUTION TIME ');
                    WriteString(OutFile,Current".MET);
                    writeln(OutFile);
                    writeln(OutFile);
                        writeln(OutFile,'END');
                    Current := Current".Link;

                        (* Dynamically create new file name *)
                    if NodeName[41] = '9' then
                        begin
                        NodeName[41] := '0';
                        NodeName[40] := succ(NodeName[40]);
                        end    (* if *)
                    else
                        NodeName[41] := succ(NodeName[41]);
                    end;    (* while *)
            end;    (* MakePSDL *)


(*------------------------------------------------------*)
```

```pascal
procedure MakeDataStream (Head: DataPtr);

    (* Generate PSDL Data Stream *)
var
   Temp: DataPtr;
   Outfile: text;

begin
   rewrite(Outfile,'/n/suns2/work/caps/prototypes/psdl.ds');
   writeln(Outfile);
   if Head <> nil then
      begin
      Temp := Head;
      write(Outfile,'DATA STREAM ');
      WriteString(Outfile,Temp^.Name);
      writeln(Outfile);
      Temp := Temp^.Link;
         while Temp <> nil do
            begin
            write(Outfile,'                 ');
            WriteString(Outfile,Temp^.Name);
            writeln(Outfile);
            Temp := Temp^.Link;
            end;    (* while *)
      writeln(Outfile);
      end;    (* if *)
   end;    (* MakeDataStream *)

(*----------------------------------------------------------------*)
```

```
begin      (* main *)
   LoadDataStructure(OpHead, OpTail, DataHead, DataTail);
   MakePSDL(OpHead);
   MakeDataStream(DataHead);
end.
```

APPENDIX F ICON FOR GRAPHIC EDITOR

```
/*----------------------------------------------------------------------
/* file:     editor.icon
/* purpose:  icon for graphic editor
/* author:   roger thorstenson
/* date:     dec 1989
/*----------------------------------------------------------------------

/* Format_version=1, Width=64, Height=64, Depth=1, Valid_bits_per_item=16
 */
        0xFFFF,0xFFFF,0xFFFF,0xFFFF,0x8000,0x0000,0x0000,0x0001,
        0x8000,0x0000,0x0000,0x0001,0x8000,0x0000,0x0300,0x0001,
        0x8000,0x0000,0x0F00,0x0001,0x8000,0x0000,0x1B00,0x0001,
        0x8000,0x0000,0x3300,0x0001,0x8000,0x0000,0x6500,0x0001,
        0x8000,0x0000,0x8600,0x0001,0x8000,0x0000,0x8A00,0x0001,
        0x8000,0x0001,0x1200,0x0001,0x8000,0x0003,0x1600,0x0001,
        0x8000,0x0002,0x2400,0x0001,0x8000,0x0006,0x2400,0x0001,
        0x8000,0x000A,0x4800,0x0001,0x8000,0x0012,0x9800,0x0001,
        0x8000,0x0014,0xF000,0x0001,0x8000,0x0025,0xA000,0x0001,
        0x8000,0x0045,0x2000,0x0001,0x8000,0x0086,0x2000,0x0001,
        0x8000,0x008C,0x4000,0x0001,0x8000,0x008C,0x8000,0x0001,
        0x8000,0x0388,0x8000,0x0001,0x8000,0x028B,0x0000,0x0001,
        0x8000,0x0297,0x0000,0x0001,0x8000,0x04BA,0x0000,0x0001,
        0x8000,0x04A2,0x0000,0x0001,0x8000,0x08C4,0x0000,0x0001,
        0x8000,0x08C8,0x0000,0x0001,0x8000,0x1888,0x0000,0x0001,
        0x8000,0x1990,0x0000,0x0001,0x8000,0x3930,0x0000,0x0001,
        0x8000,0x2A60,0x0000,0x0001,0x8000,0x2BE0,0x0000,0x0001,
        0x8000,0x2E80,0x0000,0x0001,0x8000,0x4880,0x0000,0x0001,
        0x8000,0x4900,0x0000,0x0001,0x8000,0xD200,0x0000,0x0001,
        0x8000,0x9600,0x0000,0x0001,0x8000,0xAC00,0x003F,0xE001,
        0x8000,0xC800,0x0020,0x2001,0x8001,0x7000,0x0020,0x2001,
        0x8001,0xC000,0x03FF,0xFE01,0x8003,0xC000,0x0200,0x0201,
        0x8003,0x0000,0x02E5,0x2A01,0x8002,0x0000,0x0247,0x3201,
        0x8006,0x0000,0x02E5,0x2A01,0x800C,0x0000,0x0200,0x0201,
        0x8008,0x0000,0x03FF,0xFE01,0x8000,0x0000,0x0000,0x0001,
        0xFFFF,0xFFFF,0xFFFF,0xFFFF,0x8000,0x0000,0x0000,0x0001,
        0x8000,0x0410,0x2000,0x0001,0x8000,0x0410,0x2000,0x0001,
        0x8000,0x0400,0x2000,0x0001,0x8078,0x7470,0xF878,0xB801,
        0x8084,0x8C10,0x2084,0xC401,0x8084,0x8410,0x2084,0x8001,
        0x80FC,0x8410,0x2084,0x8001,0x8080,0x8410,0x2084,0x8001,
        0x8084,0x8C10,0x2484,0x8001,0x8078,0x7410,0x1878,0x8001,
        0x8000,0x0000,0x0000,0x0001,0xFFFF,0xFFFF,0xFFFF,0xFFFF
```

APPENDIX G SSL SPECIFICATION

```
/*---------------------------------------------------------------------

   file:      psdl.as.ssl
   purpose:   abstract syntax for psdl editor
   author:    laura j. white
   date:      14 nov 89

-------------------------------------------------------------------*/


root psdl_components;

list psdl_components;

psdl_components
    :   PsdlNil()
    |   PsdlPair(component psdl_components)
    ;


component
    :   NoComponent()
    |   Data(id type_spec type_impl)
    |   Op(id operator_spec operator_impl)
    ;


id
    :   IdNull()
    |   Id(IDENTIFIER)
    ;


operator_spec
    :   OpSpec(optional_interface optional_keywords optional_description
            optional_axioms)
    ;


type_spec
    :   TypeSpec(optional_type_declarations optional_operators
            optional_keywords optional_description optional_axioms)
    ;


optional list optional_operators;
optional_operators
    :   OpListNil()
    |   OpList(type_op_spec optional_operators)
    ;
```

```
type_op_spec
    :   TypeOpNil()
    |   TypeOpSpec(id operator_spec)
    ;


optional list optional_interface;
optional_interface
    :   InterFaceNil()
    |   InterFaceList(attribute optional_interface)
    ;


optional optional_requirements;
optional_requirements
    :   ReqmtsTraceNone()
    |   ReqmtsPrompt()
    |   ReqmtsTrace(id_list)
    ;


optional optional_keywords;
optional_keywords
    :   KeyWordsNone()
    |   KeyWordsPrompt()
    |   KeyWords(id_list)
    ;


optional optional_description;
optional_description
    :   InformalDescNone()
    |   InformalPrompt()
    |   InformalDesc(text)
    ;


optional optional_axioms;
optional_axioms
    :   FormalDescNone()
    |   FormalPrompt()
    |   FormalDesc(text)
    ;


attribute
    :   EmptyAttr()
    |   Input(input optional_requirements)
    |   Output(output optional_requirements)
    |   States(state optional_requirements)
    |   Generic(generic optional_requirements)
    |   Exceptions(exception optional_requirements)
    |   TimingInfo(optional_met optional_mcp optional_mrt optional_requirements)
    ;
```

```
input
    :   InputTypeDecl(type_decl more_optional_type_declarations)
    ;

output
    :   OutputTypeDecl(type_decl more_optional_type_declarations)
    ;

state
    :   StateTypeDecl(type_decl more_optional_type_declarations
                      exp optional_exp_list)
    ;

generic
    :   GenericTypeDecl(type_decl more_optional_type_declarations)
    ;

exception
    :   ExceptionList(id_list)
    ;

optional optional_met;
optional_met
    :   MetNone()
    |   MetPrompt()
    |   Met(time)
    ;

optional optional_mcp;
optional_mcp
    :   McpNone()
    |   McpPrompt()
    |   Mcp(time)
    ;

optional optional_mrt;
optional_mrt
    :   MrtNone()
    |   MrtPrompt()
    |   Mrt(time)
    ;

time
    :   Time(integer optional_unit)
    ;

integer
    :   IntegerNil()
    |   Integer(INTEGER)
    ;
```

```
        optional optional_unit;
        optional_unit
            :   UnitNil()
            |   UnitPrompt()
            |   UnitMs()
            |   UnitSec()
            |   UnitMin()
            |   UnitHrs()
            ;


    type_decl
            :   TypeDecl(id_list type_name)
            ;


    optional list more_optional_type_declarations;
    more_optional_type_declarations
            :   MoreDeclListNil()
            |   MoreDeclList(more_type_decl more_optional_type_declarations)
            ;


    more_type_decl
            :   MoreDeclNil()
            |   MoreTypeDecl(id_list type_name)
            ;


    optional list optional_type_declarations;
    optional_type_declarations
            :   OptDeclListNil()
            |   OptDeclList(opt_type_decl optional_type_declarations)
            ;


    opt_type_decl
            :   OptDeclNil()
            |   OptTypeDecl(id_list type_name)
            ;


    list id_list;
    id_list
            :   IdNil()
            |   IdPair(id id_list)
            ;


    type_name
            :   TypeName(id optional_generic_actuals)
            ;
```

211

```
optional optional_generic_actuals;
optional_generic_actuals
    :   GenActualNil()
    |   GenActualPrompt()
    |   GenActual(type_decl more_optional_type_declarations)
    ;


operator_impl
    :   OpImpl()
    |   OpImplPsdl(diagram optional_streams optional_timers
                   optional_control_constraints optional_description)
    |   OpImplAda(id text)
    ;


type_impl
    :   TypeImpl()
    |   TypeImplOp(type_name optional_operator_implementations)
    |   TypeImplAda(id text)
    ;


optional list optional_operator_implementations;
optional_operator_implementations
    :   TypeListNil()
    |   TypeList(type_op_impl optional_operator_implementations)
    ;


type_op_impl
    :   TypeOpImplNil()
    |   TypeOpImpl(id operator_impl)
    ;


diagram
    :   Diagram(link optional_links)
    ;


optional list optional_links;
optional_links
    :   LinkListNil()
    |   LinkList(opt_link optional_links)
    ;


opt_link
    :   OptLinkNil()
    |   OptLink(id id optional_time id)
    ;


link
    :   Link(id id optional_time id)
    ;
```

212

```
optional optional_time;
optional_time
    : OptTimeNil()
    | OptTimePrompt()
    | OptTime(time)
    ;


optional optional_streams;
optional_streams
    : StreamsNil()
    | StreamsPrompt()
    | Streams(type_decl more_optional_type_declarations)
    ;


optional optional_timers;
optional_timers
    : TimersNil()
    | TimersPrompt()
    | Timers(id_list)
    ;


optional optional_control_constraints;
optional_control_constraints
    : ControlNil()
    | ControlPrompt()
    | Control(constraint optional_constraints)
    ;


constraint
    : Constraint(id optional_triggers optional_period optional_finish
                 constraint_options)
    ;


optional list optional_constraints;
optional_constraints
    : OptConNil()
    | OptConList(optional_constraint optional_constraints)
    ;


optional_constraint
    : OptConstraintNil()
    | OptConstraint(id optional_triggers optional_period optional_finish
                    constraint_options)
    ;


optional optional_triggers;
optional_triggers
    : TriggersNone()
    | TriggersPrompt()
    | TriggersChoice(triggers_choice)
    ;
```

```
triggers_choice
    :   TriggerChoiceNil()
    |   Triggers(trigger)
    |   TriggersIf(opt_trigger predicate optional_requirements)
    ;


trigger
    :   TriggerNil()
    |   AllTrigger(id_list optional_requirements)
    |   SomeTrigger(id_list optional_requirements)
    ;


optional opt_trigger;
opt_trigger
    :   OptTriggerNil()
    |   OptTriggerPrompt()
    |   OptAllTrigger(id_list)
    |   OptSomeTrigger(id_list)
    ;


optional optional_period;
optional_period
    :   PeriodNone()
    |   PeriodPrompt()
    |   Period(time optional_requirements)
    ;


optional optional_finish;
optional_finish
    :   FinishNone()
    |   FinishPrompt()
    |   Finish(time optional_requirements)
    ;


optional list constraint_options;
constraint_options
    :   ConListNone()
    |   ConListOpts(con_opts constraint_options)
    ;


con_opts
    :   ConOptsNil()
    |   OptOutput(id_list predicate optional_requirements)
    |   OptException(id optional_predicate optional_requirements)
    |   OptTimer(timer_operation id optional_predicate optional_requirements)
    ;
```

214

```
timer_operation
    :  OptNil()
    |  Read()
    |  Reset()
    |  Start()
    |  Stop()
    ;


predicate
    :  Predicate(relation optional_boolean_relations)
    ;


optional optional_predicate;
optional_predicate
    :  OptPredicateNil()
    |  OptPredPrompt()
    |  OptPredicate(relation optional_boolean_relations)
    ;


relation
    :  RelNil()
    |  RelSimple(simple_expression)
    |  RelComplex(simple_expression relational_operator simple_expression)
    ;


boolean_relation
    :  BoolNone()
    |  AndRel(relation)
    |  OrRel(relation)
    ;


optional list optional_boolean_relations;
optional_boolean_relations
    :  RelListNil()
    |  RelList(boolean_relation optional_boolean_relations)
    ;


simple_expression
    :  SimExpNil()
    |  SimInt(sign integer optional_unit)
    |  SimReal(sign real)
    |  SimId(id)
    |  SimNotId(id)
    |  SimString(string)
    |  SimPred(predicate)
    |  SimNotPred(predicate)
    |  SimTrue()
    |  SimFalse()
    |  SimNotTrue()
    |  SimNotFalse()
    ;
```

```
optional list optional_exp_list;
optional_exp_list
    : ExpListNil()
    | ExpList(optional_exp optional_exp_list)
    ;


optional_exp
    : OptExpNil()
    | OptExpConst(constant)
    | OptExpId(id)
    | OptExpComplex(type_name id exp optional_exp_list)
    ;


exp
    : ExpNil()
    | ExpConst(constant)
    | ExpId(id)
    | ExpComplex(type_name id exp optional_exp_list)
    ;


string
    : String(text)
    ;


text
    : Text(id)
    ;


sign
    : SignNil()
    | Sign(SIGN)
    ;


real
    : RealNil()
    | P_Real(PREAL)
    ;


relational_operator
    : Rel_None()
    | Rel_Lt()
    | Rel_Gt()
    | Rel_Eq()
    | Rel_Ne()
    | Rel_Co()
    | Rel_Lte()
    | Rel_Gte()
    ;
```

```
constant
    :   ConstNone ()
    |   ConstInt (integer)
    |   ConstReal (real)
    |   ConstTrue ()
    |   ConstFalse ()
    ;
```

APPENDIX H SSL SPECIFICATION

```
/*-----------------------------------------------------------------------

  file:      psdl.up.ssl
  purpose:   unparsing rules for psdl editor
  author:    laura j. white
  date:      14 nov 89

-----------------------------------------------------------------------*/

psdl_components
    :  PsdlNil          [0:]
    |  PsdlPair         [0:^("%n")^]
    ;

component
    :  NoComponent      [^:"%n[component]"]
    |  Op               [^:"%nOPERATOR "^^^]
    |  Data             [^:"%nTYPE "^^^]
    ;

id
    :  IdNull           [0::-"<identifier>"]
    |  Id               [0::-^]
    ;

operator_spec
    :  OpSpec           [^:"%nSPECIFICATION%t"^^^^^"%b%nEND"]
    ;

type_spec
    :  TypeSpec         [^:"%nSPECIFICATION%t"^^^^^"%b%nEND"]
    ;

operator_impl
    :  OpImpl           [0:"%n[operator implementation]"]
    |  OpImplPsdl       [0:"%nIMPLEMENTATION%t"^^^^^"%b%nEND"]
    |  OpImplAda        [0:"%nIMPLEMENTATION ADA "^^"%t%n( "^" )%b%nEND"]
    ;

type_impl
    :  TypeImpl         [0:"%n[type implementation]"]
    |  TypeImplOp       [0:"%nIMPLEMENTATION%t%n"^^^"%b%nEND"]
    |  TypeImplAda      [0:"%nIMPLEMENTATION ADA "^^"%t%n( "^" )%b%nEND"]
    ;
```

```
optional_operators
    : OpListNil          (@:)
    | OpList             (@:^()@)
    ;


type_op_spec
    : TypeOpNil          (^:"\n(optional operator)")
    | TypeOpSpec         (^:"\nOPERATOR "^")
    ;


optional_interface
    : InterFaceNil       (@:)
    | InterFaceList      (@:^()@)
    ;


optional_requirements
    : ReqmtsTraceNone    (@:)
    | ReqmtsPrompt       (@:"\n(requirements)")
    | ReqmtsTrace        (@:"\nBY REQUIREMENTS\t\n"^"\b")
    ;


optional_keywords
    : KeyWordsNone       (@:)
    | KeyWordsPrompt     (@:"\n(keywords)")
    | KeyWords           (@:"\nKEYWORDS\t\n"^"\b")
    ;


optional_description
    : InformalDescNone   (@:)
    | InformalPrompt     (@:"\n(description)")
    | InformalDesc       (@:"\nDESCRIPTION\t\n("^")\b")
    ;


optional_axioms
    : FormalDescNone     (@:)
    | FormalPrompt       (@:"\n(axioms)")
    | FormalDesc         (@:"\nAXIOMS\t\n("^")\b")
    ;


attribute
    : EmptyAttr          (^:"\n(interface)")
    | Input              (^:"\nINPUT"^"\t"^"\b")
    | Output             (^:"\nOUTPUT"^"\t"^"\b")
    | States             (^:"\nSTATES"^"\t"^"\b")
    | Generic            (^:"\nGENERIC"^"\t"^"\b")
    | Exceptions         (^:"\nEXCEPTIONS\t\n"^"\b")
    | TimingInfo         (^:"^^^")
    ;
```

```
input
    :   InputTypeDecl      ["("%t%n"""%b"]
    ;

output
    :   OutputTypeDecl     [":"%t%n"""%b"]
    ;

state
    :   StateTypeDecl      [":"%t%n"""%nINITIALLY%n"""%b"]
    ;

generic
    :   GenericTypeDecl    [":"%t%n"""%b"]
    ;

exception
    :   ExceptionList      [":"]
    ;

optional_met
    :   MetNone            [@:]
    |   MetPrompt          [@:"%n(met)"]
    |   Met                [@:"%nMAXIMUM EXECUTION TIME ""]
    ;

optional_mcp
    :   McpNone            [@:]
    |   McpPrompt          [@:"%n(mcp)"]
    |   Mcp                [@:"%nMINIMUM CALLING PERIOD ""]
    ;

optional_mrt
    :   MrtNone            [@:]
    |   MrtPrompt          [@:"%n(mrt)"]
    |   Mrt                [@:"%nMAXIMUM RESPONSE TIME ""]
    ;

time
    :   Time               [":""]
    ;

integer
    :   IntegerNil         [@::-"<integer>"]
    |   Integer            [@::-^]
    ;

optional_unit
    :   UnitNil            [@:]
    |   UnitPrompt         [@:" (units)"]
    |   UnitMs             [@:" MS"]
```

```
          |  UnitSec          (@:" SEC")
          |  UnitMin          (@:" MIN")
          |  UnitHrs          (@:" HOURS")
          ;


type_decl
          :  TypeDecl         (^:^" : "^)
          ;


more_optional_type_declarations
          :  MoreDeclListNil  (@:)
          |  MoreDeclList     (@:^()@)
          ;


more_type_decl
          :  MoreDeclNil      (^:",\n(more type decls)")
          |  MoreTypeDecl     (^:",\n"^" : "^)
          ;


optional_type_declarations
          :  OptDeclListNil   (@:)
          |  OptDeclList      (@:^(",")@)
          ;


opt_type_decl
          :  OptDeclNil       (^:"\n(optional type decl)")
          |  OptTypeDecl      (^:"\n"^" : "^)
          ;


id_list
          :  IdNil            (@::-)
          |  IdPair           (@::-^(", ")@)
          ;


type_name
          :  TypeName         (^:^^)
          ;


optional_generic_actuals
          :  GenActualNil     (@:)
          |  GenActualPrompt  (@:" (generic actual parameters)")
          |  GenActual        (@:"[^^"]")
          ;


optional_operator_implementations
          :  TypeListNil      (@:)
          |  TypeList         (@:^()@)
          ;


type_op_impl
          :  TypeOpImplNil    (^:"\n(operator implementation)")
```

221

```
    |  TypeOpImpl        (^:"%nOPERATOR "^^)
    ;

diagram
    :  Diagram          (^:"%nGRAPH%t%n"^^"%b")
    ;

optional_links
    :  LinkListNil       [@:]
    |  LinkList          [@:^[]@]
    ;

opt_link
    :  OptLinkNil        [^:"%n(optional link)"]
    |  OptLink           [^:"%n"^"."^^"->"^]
    ;

link
    :  Link              (^:^"."^^"->"^)
    ;

optional_time
    :  OptTimeNil        [@:]
    |  OptTimePrompt     [@:" (time) "]
    |  OptTime           [@:":"^]
    ;

optional_streams
    :  StreamsNil        [@:]
    |  StreamsPrompt     [@:"%n(data stream)"]
    |  Streams           [@:"%nDATA STREAM%t%n"^^"%b"]
    ;

optional_timers
    :  TimersNil         [@:]
    |  TimersPrompt      [@:"%n(timer)"]
    |  Timers            [@:"%nTIMER%t%n"^^"%b"]
    ;

optional_control_constraints
    :  ControlNil        [@:]
    |  ControlPrompt     [@:"%n(control constraints)"]
    |  Control           [@:"%nCONTROL_CONSTRAINTS%t"^^"%b"]
    ;

constraint
    :  Constraint        [^:"%nOPERATOR %t"^^^^"%b"]
    ;

optional_constraints
    :  OptConNil         [@:]
```

222

```
        |  OptConList        [@:^[]@]
        ;


optional_constraint
        :  OptConstraintNil (^:]
        |  OptConstraint    [^:"%nOPERATOR %t"^^^^^%b"]
        ;


optional_triggers
        :  TriggersNone      [@:]
        |  TriggersPrompt    [@:"%n(triggers)"]
        |  TriggersChoice    [@:"%nTRIGGERED "^]
        ;


triggers_choice
        :  TriggerChoiceNil [@:"(trigger choice)"]
        |  Triggers          [@:^]
        |  TriggersIf        [@:^" IF "^^]
        ;


trigger
        :  TriggerNil       [@:"(trigger)"]
        |  AllTrigger       [@:"BY ALL "^^]
        |  SomeTrigger      [@:"BY SOME "^^]
        ;


opt_trigger
        :  OptTriggerNil    [@:]
        |  OptTriggerPrompt [@:"(trigger by) "]
        |  OptAllTrigger    [@:"BY ALL "^]
        |  OptSomeTrigger   [@:"BY SOME "^]
        ;


optional_period
        :  PeriodNone       [@:]
        |  PeriodPrompt     [@:"%n(period)"]
        |  Period           [@:"%nPERIOD "^^]
        ;


optional_finish
        :  FinishNone       [@:]
        |  FinishPrompt     [@:"%n(finish within)"]
        |  Finish           [@:"%nFINISH WITHIN "^^]
        ;


constraint_options
        :  ConListNone      [@:]
        |  ConListOpts      [@:^[]@]
        ;
```

```
con_opts
    :   ConOptsNil         [^:"%n(constraint options)"]
    |   OptOutput          [^:"%nOUTPUT "^" IF "^^]
    |   OptException       [^:"%nEXCEPTION "^^^]
    |   OptTimer           [^:%n"^^ "^^^]
    ;


timer_operation
    :   OptNil             [@:"(timer operation)"]
    |   Read               [@:"READ TIMER"]
    |   Reset              [@:"RESET TIMER"]
    |   Start              [@:"START TIMER"]
    |   Stop               [@:"STOP TIMER"]
    ;


predicate
    :   Predicate          [^:^^]
    ;


optional_predicate
    :   OptPredicateNil     [@:]
    |   OptPredPrompt       [@:" (IF predicate) "]
    |   OptPredicate        [@:" IF "^^]
    ;


relation
    :   RelNil             [@:"(relation)"]
    |   RelSimple          [@:^]
    |   RelComplex         [@:"%t%n"^^^"%b"]
    ;


boolean_relation
    :   BoolNone           [^:" (boolean relation) "]
    |   AndRel             [^:" AND "^]
    |   OrRel              [^:" OR "^^]
    ;


optional_boolean_relations
    :   RelListNil         [@:]
    |   RelList            [@:^^]
    ;


simple_expression
    :   SimExpNil          [@:"(simple expression)"]
    |   SimInt             [@:^^^]
    |   SimReal            [@:^^]
    |   SimId              [@:^]
    |   SimNotId           [@:"NOT "^]
    |   SimString          [@:^]
    |   SimPred            [@:^]
    |   SimNotPred         [@:"NOT "^]
```

224

```
        |   SimTrue              [@:"TRUE"]
        |   SimFalse             [@:"FALSE"]
        |   SimNotTrue           [@:"NOT TRUE"]
        |   SimNotFalse          [@:"NOT FALSE"]
        ;

optional_exp_list
        :   ExpListNil           [@:]
        |   ExpList              [@:^[]@]
        ;

optional_exp
        :   OptExpNil            [^:",%n(exp)"]
        |   OptExpConst          [^:",%n"^]
        |   OptExpId             [^:",%n"^]
        |   OptExpComplex        [^:",%n"^".""^%n("^^")"]
        ;

exp
        :   ExpNil               [@:"(exp)"]
        |   ExpConst             [@:^]
        |   ExpId                [@:^]
        |   ExpComplex           [@:^"."""^%n("^^")"]
        ;

string
        :   String               [^::="
        ;

text
        :   Text                 [^:@]
        ;

sign
        :   SignNil              [@::="<sign>"]
        |   Sign                 [@::=^]
        ;

real
        :   RealNil              [@::="<real>"]
        |   P_Real               [@::=@]
        ;

relational_operator
        :   Rel_None             [@:" [relational operator] "]
        |   Rel_Lt               [@:" < "]
        |   Rel_Gt               [@:" > "]
        |   Rel_Eq               [@:" = "]
        |   Rel_Ne               [@:" /= "]
        |   Rel_Co               [@:" : "]
        |   Rel_Lte              [@:" <= "]
```

```
    |  Rel_Gte           (@:" >= ")
    ;

constant
    :  ConstNone        (@:"[constant]")
    |  ConstInt         (@:^)
    |  ConstReal        (@:^)
    |  ConstTrue        (@:"TRUE")
    |  ConstFalse       (@:"FALSE")
    ;
```

APPENDIX I SSL SPECIFICATION

```
/*----------------------------------------------------------------------

   file:       psdl.lex.ssl
   purpose:    lexical rules for psdl editor
   author:     laura j. white
   date:       13 nov 89


------------------------------------------------------------------------*/

IDENTIFIER: IdentLex< [a-zA-Z][a-zA-Z_0-9]* >;
WHITESPACE: WhitespaceLex< [ 0 >;
INTEGER:    IntegerLex< [0-9]* >;
PREAL:      PRealLex< [0-9]*"."[0-9]* >;
SIGN:       SignLex< [-+] >;
```

APPENDIX J SSL SPECIFICATION

```
/*---------------------------------------------------------------------

   file:       psdl.ad.ssl
   purpose:    attribute declarations for psdl editor
   author:     laura j. white
   date:       13 nov 89

   --------------------------------------------------------------------*/

Ident                (synthesized id t;);
Id_list              (synthesized id_list t;);
PSDL_sign            (synthesized sign t;);
PSDL_int             (synthesized integer t;);
PSDL_real            (synthesized real t;);


id                   ~ Ident.t;
id_list              ~ Id_list.t;
sign                 ~ PSDL_sign.t;
integer              ~ PSDL_int.t;
real                 ~ PSDL_real.t;
```

APPENDIX K SSL SPECIFICATION

```
/*------------------------------------------------------------------------

   file:      psdl.ci.ssl
   purpose:   concrete input syntax for psdl editor
   author:    laura j. white
   date:      13 nov 89


   -----------------------------------------------------------------------*/

   Ident        ::= (IDENTIFIER)
                    (Ident.t = Id(IDENTIFIER););

   Id_list      ::= (Ident)
                    (Id_list.t = (Ident.t::IdNil);)
                  | (Ident ',' Id_list)
                    (Id_list$1.t = (Ident.t::Id_list$2.t););

   PSDL_sign    ::= (SIGN)
                    (PSDL_sign.t = Sign(SIGN););

   PSDL_int     ::= (INTEGER)
                    (PSDL_int.t = Integer(INTEGER););

   PSDL_real    ::= (PREAL)
                    (PSDL_real.t = P_Real(PREAL););
```

APPENDIX L SSL SPECIFICATION

```
/*--------------------------------------------------------------------

 file:         psdl.tt.ssl
 purpose:      template transformations for psdl editor
 author:       laura j. white
 date:         12 nov 89

 ----------------------------------------------------------------------*/


transform component

    on "type"
        <component> :
        Data(<id>,<type_spec>,<type_impl>),

    on "operator"
        <component> :
        Op(<id>,<operator_spec>,<operator_impl>);


transform type_op_spec

    on "enter_operator"
        <type_op_spec> :
        TypeOpSpec(<id>,<operator_spec>);


transform attribute

    on "input"
        <attribute> :
        Input(<input>,<optional_requirements>),

    on "output"
        <attribute> :
        Output(<output>,<optional_requirements>),

    on "states"
        <attribute> :
        States(<state>,<optional_requirements>),

    on "generic"
        <attribute> :
```

```
                    Generic(<generic>,<optional_requirements>),

        on "exceptions"
            <attribute> :
            Exceptions(<exception>, <optional_requirements>),

        on "timing_info"
            <attribute> :
            TimingInfo(<optional_met>,<optional_mcp>, <optional_mrt>,
                      <optional_requirements>);


transform more_type_decl

        on "enter_type_declaration"
            <more_type_decl> :
            MoreTypeDecl(<id_list>, <type_name>);


transform opt_type_decl

        on "enter_type_declaration"
            <opt_type_decl> :
            OptTypeDecl(<id_list>, <type_name>);


transform optional_generic_actuals
        on "enter_generic_actual_parameters"
            <optional_generic_actuals> :
            GenActual(<type_decl>,<more_optional_type_declarations>);


transform optional_requirements

        on "enter_requirements"
            <optional_requirements> :
            ReqmtsTrace(<id_list>);


transform optional_keywords

        on "enter_keywords"
            <optional_keywords> :
            KeyWords(<id_list>);


transform optional_description

        on "enter_description"
            <optional_description> :
            InformalDesc(<text>);
```

```
transform optional_axioms

    on "enter_axioms"
        <optional_axioms> :
        FormalDesc(<text>);


transform optional_time

    on "enter_time"
        <optional_time> :
        OptTime(<time>);


transform optional_met

    on "enter_MET"
        <optional_met> :
        Met(<time>);


transform optional_mcp

    on "enter_MCP"
        <optional_mcp> :
        Mcp(<time>);


transform optional_mrt

    on "enter_MRT"
        <optional_mrt> :
        Mrt(<time>);

transform optional_unit

    on "milliseconds"
        <optional_unit> :
        UnitMs(),

    on "seconds"
        <optional_unit> :
        UnitSec(),

    on "minutes"
        <optional_unit> :
        UnitMin(),

    on "hours"
        <optional_unit> :
        UnitHrs();
```

232

```
transform optional_requirements

    on "enter_requirements"
        <optional_requirements> :
        ReqmtsTrace(<id_list>);


transform optional_streams

    on "enter_streams"
        <optional_streams> :
        Streams(<type_decl>, <more_optional_type_declarations>);


transform optional_timers

    on "enter_timers"
        <optional_timers> :
        Timers(<id_list>);


transform optional_control_constraints

    on "enter_control_constraints"
        <optional_control_constraints> :
        Control(<constraint>, <optional_constraints>);


transform optional_constraint

    on "enter_constraint"
        <optional_constraint> :
        OptConstraint(<id>, <optional_triggers>,<optional_period>,
                    <optional_finish>, <constraint_options>);


transform operator_impl

    on "psdl_implementation"
        <operator_impl> :
        OpImplPsdl(<diagram>,<optional_streams>,<optional_timers>,
                    <optional_control_constraints>,<optional_description>),

    on "ada_implementation"
        <operator_impl> :
        OpImplAda(<id>,<text>);
```

233

```
transform type_impl

    on "padl_implementation"
        <type_impl> :
        TypeImplOp(<type_name>,<optional_operator_implementations>),

    on "ada_implementation"
        <type_impl> :
        TypeImplAda(<id>,<text>);


transform optional_triggers

    on "enter_triggers"
        <optional_triggers> :
        TriggersChoice(<triggers_choice>);


transform triggers_choice

    on "simple_triggers"
        <triggers_choice> :
        Triggers(<trigger>),

    on "triggers_with_if_predicate"
        <triggers_choice>:
        TriggersIf(<opt_trigger>,<predicate>,<optional_requirements>);


transform trigger

    on "all"
        <trigger> :
        AllTrigger(<id_list>,<optional_requirements>),

    on "some"
        <trigger> :
        SomeTrigger(<id_list>,<optional_requirements>);


transform opt_trigger

    on "all"
        <opt_trigger> :
        OptAllTrigger(<id_list>),

    on "some"
        <opt_trigger> :
        OptSomeTrigger(<id_list>);
```

```
transform optional_period

    on "enter_period"
        <optional_period> :
        Period(<time>,<optional_requirements>);


transform optional_finish

    on "enter_finish_within"
        <optional_finish> :
        Finish(<time>,<optional_requirements>);


transform con_opts

    on "output"
        <con_opts> :
        OptOutput(<id_list>,<predicate>,<optional_requirements>),

    on "exception"
        <con_opts> :
        OptException(<id>,<optional_predicate>,<optional_requirements>),

    on "timer"
        <con_opts> :
        OptTimer(<timer_operation>,<id>,<optional_predicate>,
                 <optional_requirements>);


transform optional_predicate

    on "enter_predicate"
        <optional_predicate> :
        OptPredicate(<relation>,<optional_boolean_relations>);


transform timer_operation

    on "read_timer"
        <timer_operation> :
        Read(),

    on "reset_timer"
        <timer_operation> :
        Reset(),

    on "start_timer"
        <timer_operation> :
        Start(),
```

```
on "stop_timer"
    <timer_operation> :
    Stop();


transform relation

    on "simple"
        <relation> :
        RelSimple(<simple_expression>),

    on "complex"
        <relation> :
        RelComplex(<simple_expression>,<relational_operator>,<simple_expression>);


transform simple_expression

    on "integer_exp"
        <simple_expression> :
        SimInt(<sign>,<integer>,<optional_unit>),

    on "real_exp"
        <simple_expression> :
        SimReal(<sign>,<real>),

    on "id_exp"
        <simple_expression> :
        SimId(<id>),

    on "not_id_exp"
        <simple_expression> :
        SimNotId(<id>),

    on "string_exp"
        <simple_expression> :
        SimString(<string>),

    on "predicate_exp"
        <simple_expression> :
        SimPred(<predicate>),

    on "not_predicate_exp"
        <simple_expression> :
        SimNotPred(<predicate>),

    on "true"
        <simple_expression> :
        SimTrue(),

    on "not_true"
```

```
                    <simple_expression> :
                    SimNotTrue(),

            on "false"
                    <simple_expression> :
                    SimFalse(),

            on "not_false"
                    <simple_expression> :
                    SimNotFalse();


    transform relational_operator

            on "<"
                    <relational_operator> :
                    Rel_Lt(),

            on "<="
                    <relational_operator> :
                    Rel_Lte(),

            on ">"
                    <relational_operator> :
                    Rel_Gt(),

            on ">="
                    <relational_operator> :
                    Rel_Gte(),

            on "="
                    <relational_operator> :
                    Rel_Eq(),

            on "/="
                    <relational_operator> :
                    Rel_Ne(),

            on ":"
                    <relational_operator> :
                    Rel_Co();


    transform exp

            on "constant"
                    <exp> :
                    ExpConst(<constant>),

            on "id"
                    <exp> :
```

```
                ExpId(<id>),

        on "complex"
            <exp> :
            ExpComplex(<type_name>,<id>,<exp>,<optional_exp_list>);


transform optional_exp

    on "constant"
        <optional_exp> :
        OptExpConst(<constant>),

    on "id"
        <optional_exp> :
        OptExpId(<id>),

    on "complex"
        <optional_exp> :
        OptExpComplex(<type_name>,<id>,<exp>,<optional_exp_list>);


transform constant

    on "integer"
        <constant> :
        ConstInt(<integer>),

    on "real"
        <constant> :
        ConstReal(<real>),

    on "true"
        <constant> :
        ConstTrue(),

    on "false"
        <constant> :
        ConstFalse();


transform opt_link

    on "enter_link"
        <opt_link> :
        OptLink(<id>,<id>,<optional_time>,<id>);


transform type_op_impl

    on "enter_operator_implementation"
```

```
        <type_op_impl> :
        TypeOpImpl(<id>,<operator_impl>);


transform boolean_relation

    on "and_relation"
        <boolean_relation> :
        AndRel(<relation>),

    on "or_relation"
        <boolean_relation> :
        OrRel(<relation>);
```

APPENDIX M KODIYAK TRANSLATOR SPECIFICATION

```
|-----------------------------------------------------------------
|
| File:          translator.k
| Author:        charlie altizer
| Date:          dec 88
| Last Modified: dec 89 by laura j. white
|
|-----------------------------------------------------------------


|definitions of lexical classes

%define Digit    :[0-9]
%define Int      :(Digit)+
%define Letter   :[a-zA-Z_]
%define Alpha    :((Letter)|(Digit))
%define Blank    :[ \n]
%define Char     :[^()]
%define Quote    :["]


| definitions of white space

                 :(Blank)+


| definitions of compound symbols and keywords

GTE              :">="
LTE              :"<="
NEQV             :"/="
ARROW            :"->"
TYPE             :type|TYPE
OPERATOR         :operator|OPERATOR
SPECIFICATION    :specification|SPECIFICATION
END              :end|END
GENERIC          :generic|GENERIC
INPUT            :input|INPUT
OUTPUT           :output|OUTPUT
STATES           :states|STATES
INITIALLY        :initially|INITIALLY
EXCEPTIONS       :exceptions|EXCEPTIONS
NORMAL           :normal:NORMAL
MAX_EXEC_TIME    :maximum[ ]execution[ ]time|MAXIMUM[ ]EXECUTION[ ]TIME
MAX_RESP_TIME    :maximum[ ]response[ ]time|MAXIMUM[ ]RESPONSE[ ]TIME
MIN_CALL_PERIOD  :minimum[ ]calling[ ]period|MINIMUM[ ]CALLING[ ]PERIOD
```

```
MICROSEC          :microsec|MICROSEC
MS                :ms|MS
SEC               :sec|SEC
MIN               :min|MIN
HOURS             :hours|HOURS
BY                :by( )requirements|BY( )REQUIREMENTS
KEYWORDS          :keywords|KEYWORDS
DESCRIPTION       :description|DESCRIPTION
AXIOMS            :axioms|AXIOMS
IMPLEMENTATION    :implementation|IMPLEMENTATION
ADA               :ada|Ada|ADA
GRAPH             :graph|GRAPH
DATA_STREAM       :data( )stream|DATA( )STREAM
TIMER             :timer|TIMER
CONTROL           :control( )constraints|CONTROL( )CONSTRAINTS
TRIGGERED         :triggered|TRIGGERED
ALL               :by( )all|BY( )ALL
SOME              :by( )some|BY( )SOME
PERIOD            :period|PERIOD
FINISH            :finish( )within|FINISH( )WITHIN
EXCEPTION         :exception|EXCEPTION
READ              :read( )timer|READ( )TIMER
RESET             :reset( )timer|RESET( )TIMER
START             :start( )timer|START( )TIMER
STOP              :stop( )timer|STOP( )TIMER
IF                :if|IF
NOT               :"~"|"not"|"NOT"
AND               :"&"|"and"|"AND"
OR                :"|"|"or"|"OR"
TRUE              :true|TRUE
FALSE             :false|FALSE
ID                :(Letter)(Alpha)*
STRING_LITERAL    :(Quote)(Char)*(Quote)
INTEGER_LITERAL   :(Int)
REAL_LITERAL      :(Int)"."(Int)
TEXT              :"("(Char)*")"


| operator precedence
| %left means group and evaluate from the left

%left    OR;
%left    AND;
%left    NOT;
%left    '<', '>', '=', GTE, LTE, NEQV;
%left    ':';


%%
```

241

```
| attribute declarations for nonterminal symbols

start ( trn: string; );
psdl ( trn: string;
       uncond_output_map:string->string;
       out_env:string->string;
       in_env:string->string; );
component ( trn: string;
            uncond_output_map_in:string->string;
            uncond_output_map_out:string->string;
            in_env:string->string;
            out_env:string->string; );

data_type ( trn: string;
            in_env:string->string; );
operator ( trn: string;
           uncond_output_map_in:string->string;
           uncond_output_map_out:string->string;
           in_env:string->string;
           out_env:string->string; );

type_spec ( trn: string;
            in_env:string->string; );
type_decl_1_list ( trn: string;
                   in_env:string->string; );
type_decl ( trn: string;
            in_env,out_env :string->string;
            opid:string;
            action_code:string;
            ucond_output:string; );

op_spec_0_list ( trn: string;
                 in_env:string->string; );

operator_spec ( opid:string;
                ds_decl:string;
                state_decl:string;
                ucond_output:string;
                excp_decl:string;
                in_env,out_env :string->string; )

interface ( in_env,out_env :string->string;
            in_parm, out_parm : string;
            ds_decl: string;
            state_decl:string;
            excp_decl:string;
            ucond_output:string;
            opid:string; );

attribute ( ds_decl: string;
            in_env,out_env :string->string;
```

```
                in_parm, out_parm: string;
                opid:string;
                state_decl:string;
                ucond_output:string;
                excp_decl:string; );

time ( trn: string; );
unit ( value: int; );
id_list ( trn: string;
            action_code:string;
            tname:string;
            opid:string;
            ucond_output:string;
            count : int;
            exp_env:int->string;
            in_env:string->string;
            out_env:string->string; );


reqmts_trace ( trn: string; );
functionality ( trn: string; );
keywords ( trn: string; );
informal_desc ( trn: string; );
formal_desc ( trn: string; );
type_impl ( trn: string; );
op_impl_0_list ( trn: string; );
operator_impl ( trn: string;
                out_env:string->string;
                in_env:string->string;
                uncond_output_map:string->string;
                loc_ds_decl:string;
                timer_decl:string;
                opid:string; );
psdl_impl ( trn: string;
            parent : string;
            uncond_output_map:string->string;
            in_env:string->string;
            out_env:string->string;
            loc_ds_decl:string;
            timer_decl:string; );


data_flow_diagram ( trn: string;
                    in_env, decl_map : string-> string; );


link_0_list ( trn: string;
                in_env,in_decls,out_decls : string->string; );


link ( trn: string;
        in_env,in_decls,out_decls : string->string;   );


opt_time ( trn: string; );
type_name ( trn: string; );
```

```
timers ( trn: string; );
control_constraints ( trn: string;
                      parent : string;
                      uncond_output_map:string->string;
                      in_env:string->string;
                      out_env:string->string;
                      decl_map : string->string;
                    );
constraint_options ( trn : string;
                     in_env: string->string;
                     out_env:string->string;
                     opid:string;
                    );
more_constraints (trn : string;
                  parent:string;
                  uncond_output_map:string->string;
                  in_env:string->string;
                  out_env:string->string;
                  decl_map : string->string;
                 );
opt_trig ( out_env:string->string;
           in_env: string->string;
           streams_check:string;
           end_if_streams:string;
           pred:string;
           end_if_pred:string;
         );

trigger (if: string;
         end_if:string;
         in_env, out_env:string->string; );
opt_per ( trn: string; );
opt_fin_w ( trn: string; );
streams (trn: string;
         in_env,out_env :string->string; );
timer_op ( trn: string; );
opt_if_predicate ( if: string;
                   end_if:string;
                   parent:string;
                   in_env : string-> string;  );
predicate ( trn: string;
            in_env: string->string;
            type: string; );
expression_list ( trn: string;
                  count:int;
                  exp_env:int->string; );
expression ( trn: string; );
relation (trn. string;
          in_env: string->string;
          type: string; );
simple_expression ( trn: string;
```

```
                    parent: string;
                    in_env: string->string;
                    type:string; );
rel_op (trn: string;
        left_op:string;
        right_op: string;
        opn_type: string;
        parent: string; );
sign (trn: string; );

!attrbute declarations for terminal symbols

ID( %text: string; );
TEXT( %text: string; );
STRING_LITERAL( %text: string; );
INTEGER_LITERAL( %text: string; );
REAL_LITERAL (%text: string; );

%%
!psdl grammar

start
    : psdl
     ( %output(["with PSDL_SYSTEM;\nuse PSDL_SYSTEM;\npackage TL is\n",
                  psdl.trn,"end TL;\n"]);
            psdl.in_env =  psdl.out_env;
          )
    ;


psdl
    : component psdl
      ( psdl[1].trn = [component.trn,"\n",psdl[2].trn];
            psdl[1].out_env = component.out_env +| psdl[2].out_env;
            psdl[1].uncond_output_map = component.uncond_output_map_out
               +| psdl[2].uncond_output_map;
            component.in_env = psdl[1].in_env;
            component.uncond_output_map_in = psdl[2].uncond_output_map;
            psdl[2].in_env = psdl[1].in_env ;
         )
    |
      ( psdl.trn = "";
            psdl.out_env = ((?:string:""));
            psdl.uncond_output_map = ((?:string:"")) ;
         )
    ;


component
    : data_type
      ( component.trn = "";
            component.out_env = ((?:string:""));
            component.uncond_output_map_out = ((?:string:""));
```

```
                data_type.in_env = component.in_env;
              )


        | operator
          ( component.trn = operator.trn;
                component.out_env = operator.out_env;
                component.uncond_output_map_out = operator.uncond_output_map_out;
                operator.in_env = component.in_env;
                operator.uncond_output_map_in = component.uncond_output_map_in;
              )
        ;


data_type
      : TYPE ID type_spec type_impl
         ( data_type.trn = "";
               type_spec.in_env = data_type.in_env;
             )
        ;


operator
      : OPERATOR  ID operator_spec operator_impl
         ( operator.trn =
               (operator.in_env(ID.%text^"CONSTRUCT") == "composite_operator"
                -> ["\npackage ",
                    ID.%text,"_SPEC is\n",operator_spec.ds_decl,"\n",
                    operator_impl.loc_ds_decl,"\n",
                    operator_spec.state_decl,"\n",
                    operator_impl.timer_decl,"\n",
                    operator_spec.excp_decl,"\nend ",
                    ID.%text,"_SPEC;\n",operator_impl.trn]
                #  ""
               );
           operator.uncond_output_map_out =
             ((ID.%text:operator_spec.ucond_output));

           operator_spec.opid = ID.%text;
           operator_spec.in_env = operator.in_env;
           operator_impl.opid = ID.%text;
           operator_impl.in_env = (("PARENT":ID.%text)) +| operator.in_env;
           operator_impl.uncond_output_map = operator.uncond_output_map_in;
           operator.out_env = operator_spec.out_env +| operator_impl.out_env;
             )
        ;


type_spec
      : SPECIFICATION type_decl_1_list op_spec_0_list functionality END
         ( type_spec.trn =  "";
               type_decl_1_list.in_env = type_spec.in_env;
               op_spec_0_list.in_env = type_spec.in_env;
             )
        ;
```

```
type_decl_1_list
    : type_decl
      ( type_decl_1_list.trn = type_decl.trn;
            type_decl.action_code = "type";
            type_decl.in_env = type_decl_1_list.in_env;
          )


    |
      (type_decl_1_list.trn = "";)
    ;


type_decl
    : id_list ':' type_name
      ( type_decl.trn = id_list.trn;
            type_decl.out_env = id_list.out_env;
            type_decl.ucond_output = id_list.ucond_output;
            id_list.in_env = type_decl.in_env;
            id_list.action_code = type_decl.action_code;
            id_list.tname = type_name.trn;
            id_list.opid = type_decl.opid;
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
          )


        | id_list ':' type_name ',' type_decl
          ( type_decl[1].trn = id_list.trn ^ type_decl[2].trn;
            type_decl[1].out_env = id_list.out_env +| type_decl[2].out_env;
            type_decl.ucond_output = id_list.ucond_output
                                    ^ type_decl[2].ucond_output;

            id_list.in_env = type_decl[1].in_env;
            id_list.action_code = type_decl[1].action_code;
            id_list.tname = type_name.trn;
            id_list.opid = type_decl[1].opid;
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
            type_decl[2].in_env = type_decl[1].in_env;
            type_decl[2].opid = type_decl[1].opid;
            type_decl[2].action_code = type_decl[1].action_code;
          )
    ;


op_spec_0_list
    : op_spec_0_list OPERATOR ID operator_spec
      ( op_spec_0_list[1].trn = "";
            operator_spec.in_env = op_spec_0_list.in_env;
            op_spec_0_list[2].in_env = op_spec_0_list[1].in_env;
          )
    |
      ( op_spec_0_list.trn = ""; )
    ;
```

247

```
operator_spec
    : SPECIFICATION interface functionality END
            ( operator_spec.ds_decl = interface.ds_decl;
              operator_spec.state_decl = interface.state_decl;
              operator_spec.excp_decl = interface.excp_decl;
              operator_spec.ucond_output = interface.ucond_output;
              operator_spec.out_env =
                 (interface.out_env(operator_spec.opid~"INPARM") == "" ||
                 interface.out_env(operator_spec.opid~"OUTPARM") == ""
                 -> (((operator_spec.opid~"PROCCALL"):
                        (interface.out_env(operator_spec.opid~"INPARM"),
                         interface.out_env(operator_spec.opid~"OUTPARM")}))
                    # (((operator_spec.opid~"PROCCALL"):
                        (interface.out_env(operator_spec.opid~"INPARM"),",",
                         interface.out_env(operator_spec.opid~"OUTPARM")}))
                 ) +| interface.out_env;

              interface.in_env = operator_spec.in_env;
              interface.opid = operator_spec.opid;
            )
        ;


interface
    : interface attribute reqmts_trace
        ( interface[1].ds_decl =
                (interface[2].ds_decl,"\n",attribute.ds_decl};
          interface[1].state_decl =
                (interface[2].state_decl,"\n",attribute.state_decl};
          interface[1].excp_decl =
                (interface[2].excp_decl,"\n",attribute.excp_decl};
          interface[1].in_parm = interface[2].in_parm ~ attribute.in_parm;
          interface[1].out_parm = interface[2].out_parm ~ attribute.out_parm;

          interface[1].out_env =
             (((interface[1].opid~"INPARM"):interface[1].in_parm)
             ((interface[1].opid~"OUTPARM"):interface[1].out_parm) )
             +| interface[2].out_env  +| attribute.out_env;

          interface[1].ucond_output = interface[2].ucond_output
                                        ~ attribute.ucond_output;

          interface[2].opid = interface[1].opid;
          interface[2].in_env = interface[1].in_env;
          attribute.in_env = interface[1].in_env;
          attribute.opid = interface[1].opid;
        )

    |
      (interface.ds_decl = "";
          interface.state_decl = "";
          interface.excp_decl = "";
```

248

```
                        interface.in_parm = "";
                        interface.out_parm = "";
                        interface.out_env = ((?:string:""));
                        interface.ucond_output = "";
                        )
                ;

        attribute
            : GENERIC type_decl
                    ( type_decl.action_code = "";
                      type_decl.opid = attribute.opid;
                      type_decl.in_env = attribute.in_env;
                      attribute.out_env = type_decl.out_env;
                      attribute.ds_decl = "";
                      attribute.state_decl = "";
                      attribute.excp_decl = "";
                      attribute.in_parm = "";
                      attribute.out_parm = "";
                      attribute.ucond_output = "";
                    )

            | INPUT type_decl
                    ( type_decl.action_code = "input";
                      type_decl.opid = attribute.opid;
                      type_decl.in_env = attribute.in_env;
                      attribute.out_env = type_decl.out_env;
                      attribute.ds_decl = type_decl.txn;
                      attribute.state_decl = "";
                      attribute.excp_decl = "";
                      attribute.in_parm = type_decl.out_env(attribute.opid^"INPARM");
                      attribute.out_parm = "";
                      attribute.ucond_output = "";

                    )

            | OUTPUT type_decl
                    ( type_decl.action_code = "output";
                      type_decl.opid = attribute.opid;
                      type_decl.in_env = attribute.in_env;
                      attribute.out_env = type_decl.out_env;
                      attribute.ds_decl = type_decl.txn;
                      attribute.state_decl = "";
                      attribute.excp_decl = "";
                      attribute.in_parm = "";
                      attribute.out_parm = type_decl.out_env(attribute.opid^"OUTPARM");
                      attribute.ucond_output = type_decl.ucond_output;

                    )

            | STATES id_list ':' ID INITIALLY expression_list
                    ( id_list.action_code = "states";
```

249

```
                        id_list.opid = attribute.opid;
                        id_list.tname = ID.#text;
                        id_list.count = 1;
                        id_list.exp_env = expression_list.exp_env;
                        id_list.in_env = attribute.in_env;
                        expression_list.count = 1;
                        attribute.out_env = id_list.out_env;
                        attribute.ds_decl = "";
                        attribute.state_decl = id_list.trn;
                        attribute.excp_decl = "";
                        attribute.in_parm = "";
                        attribute.out_parm = "";
                        attribute.ucond_output = "";


                )


        | EXCEPTIONS id_list
                ( id_list.action_code = "excp";
                  id_list.tname = "exception";
                  id_list.opid = attribute.opid;
                  id_list.count = 1;
                  id_list.exp_env = ((?:int:""));
                  id_list.in_env = attribute.in_env;
                  attribute.out_env = id_list.out_env;
                  attribute.ds_decl = "";
                  attribute.state_decl = "";
                  attribute.excp_decl = [id_list.trn," : PSDL_EXCEPTION;\n"];
                  attribute.in_parm = "";
                  attribute.out_parm = "";
                  attribute.ucond_output = "";


                )


        | MAX_EXEC_TIME time
                ( attribute.ds_decl = "";
                  attribute.state_decl = "";
                  attribute.excp_decl = "";
                  attribute.out_env = ((?:string:""));
                  attribute.in_parm = "";
                  attribute.out_parm = "";
                  attribute.ucond_output = "";


                )

        | MIN_CALL_PERIOD time
                ( attribute.ds_decl = "";
                  attribute.state_decl = "";
                  attribute.excp_decl = "";
                  attribute.out_env = ((?:string:""));
                  attribute.in_parm = "";
                  attribute.out_parm = "";
```

```
                    attribute.ucond_output = "";

              )

          | MAX_RESP_TIME time
            ( attribute.ds_decl = "";
              attribute.state_decl = "";
              attribute.excp_decl = "";
              attribute.out_env = ((?:string:""));
              attribute.in_parm = "";
              attribute.out_parm = "";
              attribute.ucond_output = "";

              )


    id_list
        : ID ',' id_list
          ( id_list[1].trn =
              (id_list[1].action_code == "input" ||
               id_list[1].action_code == "output"
                 -> (id_list[1].in_env(id_list[1].opid^"PARENT") == ""
                     -> ["package DS",ID.%text," is new ",
                        (id_list[1].in_env(ID.%text^"BUFF_TYPE") == "fifo"
                          -> "FIFO_BUFFER"
                          # "SAMPLED_BUFFER")
                        ,"(",id_list[1].tname,");\n"]
                     # ["package DS",ID.%text," renames ",
                        id_list[1].in_env(id_list[1].opid^"PARENT"),"_SPEC.DS",
                        ID.%text,";\n"]) ^ id_list[2].trn

              # id_list[1].action_code == "states"
                 -> ["package DS",ID.%text," is new STATE_VARIABLE(",
                     id_list[1].tname,", ",id_list[1].exp_env(id_list[1].count),
                     ");\n"] ^ id_list[2].trn

              # id_list[1].action_code == "excp"
                 -> ["EX",ID.%text,",",id_list[2].trn]

              # id_list[1].action_code == "stream"
                 -> ["package DS",ID.%text," is new ",
                     (id_list[1].in_env(ID.%text^"BUFF_TYPE") == "fifo"
                       -> "FIFO_BUFFER"
                       # "SAMPLED_BUFFER"), "(",id_list[1].tname,");\n"]
                     ^ id_list[2].trn

              # id_list[1].action_code == "timer"
                 -> ["TL",ID.%text,",",id_list[2].trn]


              # id_list[1].action_code == "by_all"
```

```
                    -> [id_list(1).opid,"_SPEC.DS",ID.%text,".NEW_DATA"," AND \n",
                       id_list(2).trn]

# id_list(1).action_code == "by_some"
        -> [id_list(1).opid,"_SPEC.DS",ID.%text,".NEW_DATA"," OR \n",
           id_list(2).trn]

# id_list(1).action_code == "co_output"
        -> [id_list(1).in_env(id_list(1).opid~"PARENT"),"_SPEC.DS",
           ID.%text,".WRITE(",ID.%text,");\n",id_list(2).trn]

# "");

id_list(2).in_env = id_list(1).in_env;
id_list(2).action_code = id_list(1).action_code;
id_list(2).opid = id_list(1).opid;
id_list(2).tname = id_list(1).tname;
id_list(2).exp_env = id_list(1).exp_env;
id_list(2).count = id_list(1).exp_env(id_list(1).count + 1) <> ""
        -> id_list(1).count + 1
        # id_list(1).count;

id_list(1).out_env =
(id_list(1).action_code == "by_all"
 -> (((ID.%text~"BUFF_TYPE"):"fifo")) +| id_list(2).out_env

 # id_list(1).action_code == "by_some"
   --> (((ID.%text~"BUFF_TYPE"):"sampled")) +| id_list(2).out_env

 # id_list(1).action_code == "input"
    -> (((ID.%text~"TYPE"):id_list(1).tname)
       ((ID.%text~"CONSTRUCT"):"data_stream")
       ((id_list(1).opid~"INPARM"):[ID.%text,",",
         id_list(2).out_env(id_list(2).opid~"INPARM")])
       ) +| id_list(2).out_env

 # id_list(1).action_code == "output"
    -> (((ID.%text~"TYPE"):id_list(1).tname)
       ((ID.%text~"CONSTRUCT"):"data_stream")
       ((id_list(1).opid~"OUTPARM"):[ID.%text,",",
         id_list(2).out_env(id_list(2).opid~"OUTPARM")])
       ) +| id_list(2).out_env

 # id_list(1).action_code == "stream"
    -> (((ID.%text~"TYPE"):id_list(1).tname)
       ((ID.%text~"CONSTRUCT"):"data_stream")) +| id_list(2).out_env

 # id_list(1).action_code == "states"
    -> (((ID.%text~"TYPE"):id_list(1).tname)
       ((ID.%text~"CONSTRUCT"):"data_stream")) +| id_list(2).out_env
```

252

```
# id_list[1].action_code == "excp"
  -> (((ID.%text^"CONSTRUCT"):"exception")) +| id_list[2].out_env

# id_list[1].action_code == "timer"
  -> (((ID.%text^"CONSTRUCT"):"timer")) +| id_list[2].out_env

# id_list[1].action_code == "co_output"
  -> (((id_list[1].opid,"_",ID.%text,"OUTPUT"):"conditional"))
     +| id_list[2].out_env

# ((?:string:""))
);

id_list.ucond_output =
((id_list[1].in_env(id_list[1].opid^"_"^ID.%text^"OUTPUT")
   <> "conditional" ) &&
 (id_list[1].action_code == "output")
    -> [id_list[1].in_env(id_list[1].opid^"PARENT"),"_SPEC.DS",
        ID.%text,".WRITE(",ID.%text,");\n"]
    # ""
 )  ^ id_list[2].ucond_output;

)


| ID
( id_list.trn =
     (id_list.action_code == "input" ||
      id_list.action_code == "output"
       -> (id_list.in_env(id_list.opid^"PARENT") == ""
           -> ["package DS",ID.%text," is new ",
               (id_list.in_env(ID.%text^"BUFF_TYPE") == "fifo"
                -> "FIFO_BUFFER"
                # "SAMPLED_BUFFER")
               ,"(",id_list.tname,");\n"]
          # ["package DS",ID.%text," renames ",
             id_list.in_env(id_list.opid^"PARENT"),"_SPEC.DS",
             ID.%text,";\n"])

     # id_list.action_code == "states"
           -> ["package DS",ID.%text," is new STATE_VARIABLE(",
              id_list.tname,", ",id_list.exp_env(id_list.count),
              ");\n"]
     # id_list.action_code == "excp"
           -> ["EX",ID.%text]

     # id_list.action_code == "stream"
        -> ["package DS",ID.%text," is new ",
            (id_list.in_env(ID.%text^"BUFF_TYPE") == "fifo"
            -> "FIFO_BUFFER"
            # "SAMPLED_BUFFER"), "(",id_list.tname,");\n"]
```

```
# id_list.action_code == "timer"
   -> ["TL",ID.%text]

# id_list.action_code == "by_all"
   -> [id_list.opid,"_SPEC.DS",ID.%text,".NEW_DATA"]

# id_list.action_code == "by_some"
   -> [id_list.opid,"_SPEC.DS",ID.%text,".NEW_DATA"]

# id_list.action_code == "co_output"
   -> [id_list.in_env(id_list.opid^"PARENT"),"_SPEC.DS",ID.%text,
        ".WRITE(",ID.%text,");\n"]

# "");

id_list.out_env =
(id_list.action_code == "by_all"
-> (((ID.%text^"BUFF_TYPE"):"fifo"))

# id_list.action_code == "by_some"
   -> (((ID.%text^"BUFF_TYPE"):"sampled"))

# id_list[1].action_code == "input"
   -> (((ID.%text^"TYPE"):id_list[1].tname)
       ((ID.%text^"CONSTRUCT"):"data_stream")
       ((id_list.opid^"INPARM"):ID.%text))

# id_list[1].action_code == "output"
   -> (((ID.%text^"TYPE"):id_list[1].tname)
       ((ID.%text^"CONSTRUCT"):"data_stream")
       ((id_list.opid^"OUTPARM"):ID.%text))

# id_list[1].action_code == "stream"
   -> (((ID.%text^"TYPE"):id_list[1].tname)
       ((ID.%text^"CONSTRUCT"):"data_stream"))

# id_list[1].action_code == "states"
   -> (((ID.%text^"TYPE"):id_list[1].tname)
       ((ID.%text^"CONSTRUCT"):"data_stream"))

# id_list.action_code == "excp"
   -> (((ID.%text^"CONSTRUCT"):"exception"))

# id_list.action_code == "timer"
   -> (((ID.%text^"CONSTRUCT"):"timer"))

# id_list.action_code == "co_output"
   -> (([id_list.opid,"_",ID.%text,"OUTPUT"]:"conditional"))

# ((?:string:""))
);
```

254

```
            id_list.ucond_output =
            ((id_list.in_env(id_list.opid^"_"^ID.%text^"OUTPUT")<>"conditional")
             && (id_list.action_code == "output")
             -> {id_list.in_env(id_list.opid^"PARENT"),"_SPEC.DS",ID.%text,
                  ".WRITE(",ID.%text,");\n"]
              # ""
            );

            )
    ;


time
    : INTEGER_LITERAL unit
      ( time.trn = ""; )
    ;

unit
    : MICROSEC
            ( unit.value = 1;
            )

    | MS
            ( unit.value = 1000;
            )

    | SEC
            ( unit.value = 1000000;
            )

    | MIN
            ( unit.value = 60000000;
            )

    | HOURS
            ( unit.value = 3600000000;
            )
    ;

reqmts_trace
    : BY id_list
      ( reqmts_trace.trn = "";
            id_list.in_env = ((?:string:""));
            id_list.action_code = "";
            id_list.tname = "";
            id_list.opid = "";
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
      )
```

```
          |
            ( reqmts_trace.trn = ""; )
            ;


functionality
     : keywords informal_desc formal_desc
       ( functionality.trn = ""; )
     ;


keywords
     : KEYWORDS id_list
       ( keywords.trn = "";
            id_list.in_env = ((?:string:""));
            id_list.action_code = "";
            id_list.tname = "";
            id_list.opid = "";
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
          )
     |
       ( keywords.trn = ""; )
     ;


informal_desc
     : DESCRIPTION TEXT
       ( informal_desc.trn = "\n"; )
     |
       (informal_desc.trn = ""; )
     ;


formal_desc
     : AXIOMS TEXT
       ( formal_desc.trn = "\n"; )
     |
       ( formal_desc.trn = ""; )
     ;


type_impl
     : IMPLEMENTATION ADA ID END
       ( type_impl.trn = ["procedure ",ID.text," is;\n"]; )
     | IMPLEMENTATION type_name op_impl_0_list END
       ( type_impl.trn = ["\n package DATA_TYPES is \n",type_name.trn,"\n",
               op_impl_0_list.trn,"\n",
               "end;\n"]; )
     ;


op_impl_0_list
     : op_impl_0_list OPERATOR  ID operator_impl
       ( op_impl_0_list[1].trn = "";
            operator_impl.opid = ID.text; )
     |
```

```
                ( op_impl_0_list[1].trn - ""; )
            ;


operator_impl
    : IMPLEMENTATION ADA ID END
        ( operator_impl.trn - "";
            operator_impl.loc_ds_decl - "";
            operator_impl.timer_decl - "";
            operator_impl.out_env - (((ID.%text""CONSTRUCT"):"atomic_operator"));
            )


    | IMPLEMENTATION psdl_impl
        ( operator_impl.trn - psdl_impl.trn;
            operator_impl.loc_ds_decl - psdl_impl.loc_ds_decl;
            operator_impl.timer_decl - psdl_impl.timer_decl;
            psdl_impl.parent - operator_impl.opid;
            psdl_impl.in_env - operator_impl.in_env;
            psdl_impl.uncond_output_map - operator_impl.uncond_output_map;
            operator_impl.out_env -
              (((operator_impl.opid""CONSTRUCT"):"composite_operator")) +|
                psdl_impl.out_env;
            )
        ;


psdl_impl
    : data_flow_diagram streams timers control_constraints informal_desc END
        ( psdl_impl.trn - control_constraints.trn;
            psdl_impl.out_env -  streams.out_env +| control_constraints.out_env;
            psdl_impl.loc_ds_decl - streams.trn;
            psdl_impl.timer_decl - timers.trn;
            data_flow_diagram.in_env - psdl_impl.in_env;
            streams.in_env - psdl_impl.in_env;
            control_constraints.parent - psdl_impl.parent;
            control_constraints.in_env - psdl_impl.in_env;
            control_constraints.decl_map - data_flow_diagram.decl_map;
            control_constraints.uncond_output_map - psdl_impl.uncond_output_map;
            )
            ;


data_flow_diagram
    : GRAPH link_0_list
        ( data_flow_diagram.trn - "";
            data_flow_diagram.decl_map - link_0_list.out_decls;
            link_0_list.in_decls - ((?:string:""));
            link_0_list.in_env - data_flow_diagram.in_env;
            )
    ;


link_0_list
    : link link_0_list
        ( link_0_list[1].trn - "";
```

257

```
                    link_0_list(1).out_decls = link_0_list(2).out_decls;
                    link_0_list(2).in_decls = link.out_decls;
                    link.in_decls = link_0_list(1).in_decls;
                    link.in_env = link_0_list(1).in_env;
                    link_0_list(2).in_env = link_0_list(1).in_env;
                    }

          |
            ( link_0_list.trn = "";
                    link_0_list.out_decls = link_0_list.in_decls;
                    }
          ;


link
      : ID '.' ID opt_time ARROW ID
        ( link.trn = "";
            link.out_decls =
                (link.in_decls(ID(3).%text^ID(1).%text^"READ") == "dup"
                -> ((?:string:""))
            # (((ID(3).%text^"READ"):(link.in_decls(ID(3).%text^"READ"),
                link.in_env("PARENT"),"_SPEC.DS",ID(1).%text,".READ(",ID(1).%text,
                ");\n")) ((ID(3).%text^ID(1).%text^"READ"):"dup"))
                ) +|

                    (link.in_decls([ID(2).%text,"_",ID(1).%text]) == "dup"
                      -> ((?:string:""))
                      # ((ID(2).%text:(link.in_decls(ID(2).%text),
                          ID(1).%text," : ",link.in_env(ID(1).%text^"TYPE"),
                          ";\n")) ((ID(2).%text,"_",ID(1).%text]:"dup"))
                    )  +|

                    (link.in_decls([ID(3).%text,"_",ID(1).%text]) == "dup"
                      -> ((?:string:""))
                      # ((ID(3).%text:(link.in_decls(ID(3).%text),
                          ID(1).%text," : ",link.in_env(ID(1).%text^"TYPE"),
                          ";\n")) ((ID(3).%text,"_",ID(1).%text]:"dup"))
                    )  +|   link.in_decls;
                }
          ;


opt_time
      : ':' time
        ( opt_time.trn = ""; )
      |
        ( opt_time.trn = "\n"; )
      ;



streams
      : DATA_STREAM type_decl
        ( streams.trn = type_decl.trn;
                streams.out_env = type_decl.out_env;
```

258

```
                    type_decl.opid = "";
                    type_decl.action_code = "stream";
                    type_decl.in_env = streams.in_env;
                )

        |
          (streams.trn = "";
              streams.out_env = ((?:string:""));
              )
        ;


type_name
    : ID '[' type_decl ']'
      ( type_name.trn = [ID.%text,"[",type_decl.trn,"]\n"];
            type_decl.opid = "";
            type_decl.action_code = "tname"; )
    | ID
      ( type_name.trn = ID.%text; )
    ;


timers
    : TIMER id_list
      ( timers.trn = [id_list.trn,"  : PSDL_TIMER;\n"];
            id_list.in_env = ((?:string:""));
            id_list.action_code = "timer";
            id_list.tname = "";
            id_list.opid = "";
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
            )


    |
        (timers.trn = "";
            )
          ;


control_constraints
    : CONTROL
      ( control_constraints.trn = "";
            control_constraints.out_env = ((?:string:""));
            )

    | CONTROL OPERATOR ID opt_trig opt_per opt_fin_w constraint_options
          more_constraints
        (control_constraints.trn =
            (control_constraints.in_env(ID.%text^"CONSTRUCT")
                == "composite_operator"
              -> ["procedure ",control_constraints.in_env("PARENT"),"_",
                  ID.%text," is\n begin\n     null;\n end ",
                  control_constraints.in_env("PARENT"),"_",ID.%text,";\n"]

              #  ["procedure ",control_constraints.in_env("PARENT"),
```

```
                          "_",ID.%text," is\n",control_constraints.decl_map(ID.%text),
                          "\nbegin\n",opt_trig.streams_check,
                          control_constraints.decl_map(ID.%text^"READ"),
                          opt_trig.pred,
                          (control_constraints.in_env(ID.%text^"PROCCALL") == ""
                           -> [ID.%text,";\n"]
                           # [ID.%text,"(",
                               control_constraints.in_env(ID.%text^"PROCCALL") ,");\n"]
                          ),
                          constraint_options.trn,"\n",
                          control_constraints.uncond_output_map(ID.%text),
                          opt_trig.end_if_pred,opt_trig.end_if_streams,
                          "end ",control_constraints.in_env("PARENT"),
                          "_",ID.%text,";\n")
                    ) ^ more_constraints.trn;

                 opt_trig.in_env = control_constraints.in_env;
                 constraint_options.in_env = control_constraints.in_env;
                 constraint_options.opid = ID.%text;
                 control_constraints.out_env =
                     (((ID.%text^"PARENT"):control_constraints.parent))
                     +| opt_trig.out_env
                     +| constraint_options.out_env
                     +| more_constraints.out_env;

                 more_constraints.parent = control_constraints.parent;
                 more_constraints.in_env = control_constraints.in_env;
                 more_constraints.uncond_output_map =
                     control_constraints.uncond_output_map;
                 more_constraints.decl_map = control_constraints.decl_map;
                 )


         |
                 (control_constraints.trn = "";
                  control_constraints.out_env = ((?:string:""));
                 )


         ;


   more_constraints
           : OPERATOR ID opt_trig opt_per opt_fin_w constraint_options
             more_constraints
             (more_constraints[1].trn =
               (more_constraints.in_env(ID.%text^"CONSTRUCT") ==
                  "composite_operator"
                -> ["procedure ",more_constraints[1].in_env("PARENT"),"_",
                    ID.%text," is\n begin\n    null;\n end ",
                    more_constraints[1].in_env("PARENT"),"_",ID.%text,";\n"]

                # ["procedure ",more_constraints[1].in_env("PARENT"),
                   "_",ID.%text," is\n",more_constraints[1].decl_map(ID.%text),
```

260

```
            "\nbegin\n",opt_trig.streams_check,
            more_constraints.decl_map(ID.%text^"READ"),
            opt_trig.pred,
            (more_constraints[1].in_env(ID.%text^"PROCCALL") == ""
             -> [ID.%text,";\n"]
             #  [ID.%text,"(",
                   more_constraints[1].in_env(ID.%text^"PROCCALL"),");\n"]
            ),
            constraint_options.trn,"\n",
            more_constraints[1].uncond_output_map(ID.%text),
            opt_trig.end_if_pred,opt_trig.end_if_streams,
            "end ",more_constraints[1].in_env("PARENT"),
            "_",ID.%text,";\n"]
       ) ^ more_constraints[2].trn;

    opt_trig.in_env = more_constraints.in_env;
    constraint_options.in_env = more_constraints.in_env;
    constraint_options.opid = ID.%text;
    more_constraints[1].out_env =
        {((ID.%text^"PARENT"):more_constraints.parent)} +|
         opt_trig.out_env +|
         constraint_options.out_env +|
         more_constraints[2].out_env;
    more_constraints[2].in_env = more_constraints[1].in_env;
    more_constraints[2].uncond_output_map =
         more_constraints[1].uncond_output_map;
    more_constraints[2].decl_map = more_constraints[1].decl_map;


    )


    |

    (more_constraints.trn = "";
     more_constraints.out_env = {(?:string:"")}  ;
    )
    ;



constraint_options
     :  OUTPUT id_list IF predicate reqmts_trace constraint_options
     ( constraint_options[1].trn =
        ["if ",predicate.trn,"\nthen\n    ",id_list.trn,"\nend if;\n",
         constraint_options[2].trn] ;

        constraint_options[2].opid = constraint_options[1].opid;
        constraint_options[2].in_env = constraint_options[1].in_env;
        constraint_options[1].out_env = id_list.out_env +|
                                        constraint_options[2].out_env;
        predicate.in_env = constraint_options[1].in_env;
        id_list.in_env = constraint_options[1].in_env;
        id_list.action_code = "co_output";
        id_list.tname = "";
```

```
                    id_list.opid = constraint_options.opid;
                    id_list.count = 1;
                    id_list.exp_env = ((?:int:""));
                )


        | EXCEPTION ID opt_if_predicate reqmts_trace constraint_options
          ( constraint_options[1].trn = constraint_options[2].trn;
            constraint_options[1].out_env = constraint_options[2].out_env;
            constraint_options[2].opid = constraint_options[1].opid;
            constraint_options[2].in_env = constraint_options[1].in_env;
            opt_if_predicate.in_env = constraint_options[1].in_env;
          )


        | timer_op ID  opt_if_predicate reqmts_trace constraint_options
          ( constraint_options[1].trn =
              (opt_if_predicate.if,timer_op.trn,"(",
               constraint_options[1].in_env("PARENT"),"_SPEC.TI.",ID.$text,
               ");\n",opt_if_predicate.end_if,constraint_options[2].trn);
            constraint_options[1].out_env = constraint_options[2].out_env;
            constraint_options[2].opid = constraint_options[1].opid;
            constraint_options[2].in_env = constraint_options[1].in_env;
            opt_if_predicate.in_env = constraint_options[1].in_env;
          )


        |
          ( constraint_options.trn = "";
            constraint_options.out_env = ((?:string:""));
          )
        ;

opt_trig
    : TRIGGERED trigger opt_if_predicate reqmts_trace
          ( opt_trig.out_env = trigger.out_env;
            opt_trig.pred = opt_if_predicate.if;
            opt_trig.end_if_pred = opt_if_predicate.end_if;
            opt_trig.streams_check = trigger.if;
            opt_trig.end_if_streams = trigger.end_if;
            trigger.in_env = opt_trig.in_env;
            opt_if_predicate.in_env = opt_trig.in_env;
          )
    |
          (opt_trig.out_env = ((?:string:"")));
           opt_trig.pred = "";
           opt_trig.end_if_pred = "";
           opt_trig.streams_check = "";
           opt_trig.end_if_streams = "";
          )
    ;

trigger
    : ALL id_list
```

262

```
        ( trigger.if = ["if ",id_list.trn,"\nthen\n"];
            trigger.end_if = "and if;\n";
            trigger.out_env = id_list.out_env;
            id_list.action_code = "by_all";
            id_list.tname = "";
            id_list.opid = trigger.in_env("PARENT");
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
        )

    | SOME id_list
      ( trigger.if = ["if ",id_list.trn,"\nthen\n"];
            trigger.end_if = "end if;\n";
            trigger.out_env = id_list.out_env;
            id_list.action_code = "by_some";
            id_list.tname = "";
            id_list.opid = trigger.in_env("PARENT");
            id_list.count = 1;
            id_list.exp_env = ((?:int:""));
        )

    |
      (trigger.if = "";
            trigger.end_if = "";
            trigger.out_env = ((?:string:""));
        )
    ;

opt_per
    :PERIOD time reqmts_trace
      ( opt_per.trn = "\n"; )
    |
      ( opt_per.trn = ""; )
    ;

opt_fin_w
    : FINISH time reqmts_trace
      ( opt_fin_w.trn = "\n"; )
    |
      ( opt_fin_w.trn = ""; )
    ;




timer_op
    : READ
          ( timer_op.trn = "PSDL_TIMER.READ"; )
        | RESET
      ( timer_op.trn = "PSDL_TIMER.RESET"; )
    | START
```

263

```
                        ( timer_op.trn = "PSDL_TIMER.START"; )
                | STOP
                        ( timer_op.trn = "PSDL_TIMER.STOP"; )
                ;


opt_if_predicate
        : IF predicate
          ( opt_if_predicate.if = ["if ",predicate.trn,"\nthen\n"];
                opt_if_predicate.end_if = "end if;\n";
                predicate.in_env = opt_if_predicate.in_env;
              )
            |
              (opt_if_predicate.if = "";
               opt_if_predicate.end_if = "";
               )
           ;


expression_list
      : expression
        (expression_list.trn = expression.trn;
             expression_list.exp_env = ((expression_list.count:expression.trn)
                                        (0:i2s(expression_list.count))
                                        (?:int:""))); )

         | expression ',' expression_list
        (expression_list[1].trn =
        [expression.trn,",",expression_list[2].trn];
             expression_list[1].exp_env =
             ((expression_list[1].count:expression.trn)) +|
             expression_list[2].exp_env;
             expression_list[2].count = expression_list[1].count + 1; )
         ;

expression
        : INTEGER_LITERAL
          (expression.trn = INTEGER_LITERAL.%text; )
        | REAL_LITERAL
          (expression.trn = REAL_LITERAL.%text; )
        | STRING_LITERAL
          (expression.trn = STRING_LITERAL.%text; )
        | TRUE
          (expression.trn = " true "; )
        | FALSE
          (expression.trn = " false "; )
        | ID
          (expression.trn = ID.%text; )
        | type_name '.' ID '(' expression_list ')'
          (expression.trn = [type_name.trn,ID.%text,"(",expression_list.trn,
                           ") "];
             expression_list.count = 1; )
```

264

```
        ;

    predicate
        : relation
          (predicate.trn = relation.trn;
           predicate.type = relation.type;
           relation.in_env = predicate.in_env;
          )

        | relation AND predicate
          (predicate[1].trn = [relation.trn," and ",predicate[2].trn];
           predicate[1].type = "";
           predicate[2].in_env = predicate[1].in_env;
           relation.in_env = predicate[1].in_env;
          )

        | relation OR predicate
          (predicate[1].trn = [relation.trn," or ",predicate[2].trn];
           predicate[1].type = "";
           predicate[2].in_env = predicate[1].in_env;
           relation.in_env = predicate[1].in_env;
          )
        ;


    relation
        : simple_expression rel_op simple_expression
          (relation.trn = rel_op.trn;
           simple_expression[1].in_env = relation.in_env;
           simple_expression[2].in_env = relation.in_env;
           relation.type =
             (simple_expression[1].type == "timer" ||
              simple_expression[2].type == "timer"
              -> "timer"
              # simple_expression[1].type == "excp" ||
                simple_expression[2].type == "excp"
                -> "excp"
                # ""
             );
           rel_op.left_op = simple_expression[1].trn;
           rel_op.right_op = simple_expression[2].trn;
           rel_op.parent = relation.in_env("PARENT");
           rel_op.opn_type =
             (simple_expression[1].type == "timer" ||
              simple_expression[2].type == "timer"
              -> "timer_op"
              # "arithmetic"
             );
          )
```

```
          | simple_expression
            (relation.trn = simple_expression.trn;
             relation.type = simple_expression.type;
             simple_expression.in_env = relation.in_env;
            )
          ;


simple_expression
          : INTEGER_LITERAL unit
            (simple_expression.trn = i2s(s2i(INTEGER_LITERAL.%text)
                                    * unit.value);
             simple_expression.type = "timer";
            )
          | sign INTEGER_LITERAL
            (simple_expression.trn = [sign.trn,INTEGER_LITERAL.%text];
             simple_expression.type = "";
            )
          | sign REAL_LITERAL
            (simple_expression.trn = [sign.trn,REAL_LITERAL.%text];
             simple_expression.type = "";
            )
          | ID
            (simple_expression.trn = ID.%text;
             simple_expression.type =
              (simple_expression.in_env(ID.%text^"CONSTRUCT") == "timer"
               -> ""
               #  simple_expression.in_env(ID.%text^"CONSTRUCT")
              );
            )
          | STRING_LITERAL
            (simple_expression.trn = STRING_LITERAL.%text;
             simple_expression.type = "";
            )
          | '(' predicate ')'
            (simple_expression.trn = ["(",predicate.trn,")"];
             simple_expression.type = predicate.type;
             predicate.in_env = simple_expression.in_env;
            )
          | NOT ID
            (simple_expression.trn = ["not ",ID.%text];
             simple_expression.type = "";
            )
          | NOT '(' predicate ')'
            (simple_expression.trn = ["not (",predicate.trn,")"];
             simple_expression.type = "";
             predicate.in_env = simple_expression.in_env;
            )
          | TRUE
            (simple_expression.trn = " true ";
             simple_expression.type = "";
            )
```

266

```
        | FALSE
          (simple_expression.trn = " false ";
           simple_expression.type = "";
          )
        | NOT TRUE
          (simple_expression.trn = [" not true "];
           simple_expression.type = "";
          )
        | NOT FALSE
          (simple_expression.trn = " not false ";
           simple_expression.type = "";
          )
        ;




rel_op
    : '<'
      (rel_op.trn =
            (rel_op.opn_type == "timer_op"
             -> ["PSDL_TIMER.
             #  [rel_op.left_op," < ",rel_op.right_op]
            );
          )
        | '>'
      (rel_op.trn =
            (rel_op.opn_type == "timer_op"
             -> ["PSDL_TIMER.
             #  [rel_op.left_op," > ",rel_op.right_op]
            );
          )

        | '='
      (rel_op.trn =
            (rel_op.opn_type == "timer_op"
             -> ["PSDL_TIMER.
             #  [rel_op.left_op," = ",rel_op.right_op]
            );
          )

        | GTE
      (rel_op.trn =
            (rel_op.opn_type == "timer_op"
             -> ["PSDL_TIMER.
             #  [rel_op.left_op," >= ",rel_op.right_op]
            );
          )

        | LTE
      (rel_op.trn =
```

267

```
                    (rel_op.opn_type == "timer_op"
                     -> ["PSDL_TIMER.
                     #  [rel_op.left_op," <= ",rel_op.right_op]
                    );
                  )

          | NEQV
        (rel_op.trn =
                (rel_op.opn_type == "timer_op"
                 -> ["PSDL_TIMER.
                 #  [rel_op.left_op," /= ",rel_op.right_op]
                );
              )

        | ':'
          (rel_op.trn =
            (rel_op.right_op == "NORMAL"
             -> [rel_op.parent,"_SPEC.DS",rel_op.left_op,".IS_NORMAL "]
             #  [rel_op.parent,"_SPEC.DS",rel_op.left_op,
                ".IS_EXCEPTION(",rel_op.right_op,") "]
            );
          )
        ;




sign
        : '+'
          (sign.trn = "+ "; )
        | '-'
          (sign.trn = "- "; )
        |
          (sign.trn = ""; )
        ;
```

268

APPENDIX N PSDL DATA TYPES

```
-----------------------------------------------------------------------
--
--  File:     psdl_system.a
--  Author:   Frank Palazzo
--  Date:     15 Dec 89
--  Modified: 16 Dec 89 by Laura J. White
--
-----------------------------------------------------------------------


with vstrings, TIMERS;
package PSDL_SYSTEM is

   type Int_list is array (1..10) of integer;

   package PSDL_STRINGS is new vstrings(50);
   subtype PSDL_EXCEPTION is PSDL_STRINGS.VSTRING;
   type    PSDL_TIMER is new TIMERS.TIMER;

   EXCEPTION_ERROR,
   BUFFER_UNDERFLOW,
   BUFFER_OVERFLOW   : exception;


   generic
   type ELEMENT_TYPE is private;

   package SAMPLED_STREAM is

     task DATA_STREAM  is
        pragma PRIORITY (10);
        entry CHECK (NEW_DATA : out BOOLEAN);
        entry GET   (VALUE : out ELEMENT_TYPE);
        entry GET   (VALUE : out PSDL_SYSTEM.PSDL_EXCEPTION);
        entry PUT   (VALUE : in  ELEMENT_TYPE);
        entry PUT   (VALUE : in  STRING);
        entry IS_EXCEPTION (NAME : in PSDL_STRINGS.VSTRING;
                            CHECK : out BOOLEAN);
        entry IS_NORMAL (CHECK : out BOOLEAN);

     end DATA_STREAM;
   end SAMPLED_STREAM;
```

269

```
  generic
  type ELEMENT_TYPE is private;

  package DATAFLOW_STREAM is

     task DATA_STREAM  is
       pragma PRIORITY (10);
       entry CHECK (NEW_DATA : out BOOLEAN);
       entry GET   (OUTVALUE : out ELEMENT_TYPE);
       entry PUT   (INVALUE : in  ELEMENT_TYPE);
     end DATA_STREAM;

     function FRESH return BOOLEAN;
  end DATAFLOW_STREAM;

  generic
  type ELEMENT_TYPE is private;
       INITIAL_VALUE  : ELEMENT_TYPE;

  package SAMPLED_STATE_VAR  is

     task DATA_STREAM  is
       pragma PRIORITY (10);
       entry CHECK (NEW_DATA : out BOOLEAN);
       entry GET   (OUTVALUE : out ELEMENT_TYPE);
       entry PUT   (INVALUE : in  ELEMENT_TYPE);
     end DATA_STREAM;
     function FRESH return BOOLEAN;
  end SAMPLED_STATE_VAR;

  generic
  type ELEMENT_TYPE is private;
       INITIAL_VALUE  :  ELEMENT_TYPE;

  package DATAFLOW_STATE_VAR is

     task DATA_STREAM  is
       pragma PRIORITY (10);
       entry CHECK (NEW_DATA : out BOOLEAN);
       entry GET   (OUTVALUE : out ELEMENT_TYPE);
       entry PUT   (INVALUE : in  ELEMENT_TYPE);
     end DATA_STREAM;

     function FRESH return BOOLEAN;
  end DATAFLOW_STATE_VAR;

end PSDL_SYSTEM;
```

```
package body PSDL_SYSTEM is
  package body SAMPLED_STREAM is

    package VSTRING renames PSDL_SYSTEM.PSDL_STRINGS;

    type DATA_STREAM_MODE is (NORMAL, EXCPTION);
    type DATA_STREAM_TOKEN (MODE : DATA_STREAM_MODE := NORMAL) is
      record
        INITIALIZED,
        NEW_DATA    : BOOLEAN := false;

        case MODE is
          when NORMAL =>
            N_VALUE  : ELEMENT_TYPE;
          when EXCPTION =>
            E_VALUE  : PSDL_SYSTEM.PSDL_EXCEPTION;
        end case;
      end record;

    task body DATA_STREAM is

      BUFFER : DATA_STREAM_TOKEN;
      TempExcp  : PSDL_SYSTEM.PSDL_EXCEPTION;

    begin
      loop
        select
          accept CHECK (NEW_DATA : out BOOLEAN) do
            NEW_DATA := BUFFER.NEW_DATA;
          end CHECK;
        or
          accept GET (VALUE : out ELEMENT_TYPE) do
            if not BUFFER.INITIALIZED then
              raise PSDL_SYSTEM.BUFFER_UNDERFLOW;
            elsif BUFFER.MODE = EXCPTION  then
              raise PSDL_SYSTEM.EXCEPTION_ERROR;
            else
              VALUE := BUFFER.N_VALUE;
              BUFFER.NEW_DATA := false;
            end if;
          end GET;
        or
          accept GET (VALUE : out PSDL_SYSTEM.PSDL_EXCEPTION) do
            if not BUFFER.INITIALIZED then
              raise PSDL_SYSTEM.BUFFER_UNDERFLOW;
            elsif BUFFER.MODE = NORMAL then
              raise PSDL_SYSTEM.EXCEPTION_ERROR;
            else
              VALUE := BUFFER.E_VALUE;
              BUFFER.NEW_DATA := false;
            end if;
```

```
           end GET;
       or
         accept PUT (VALUE : in ELEMENT_TYPE) do
           if (BUFFER.MODE = EXCPTION)  and BUFFER.NEW_DATA then
             raise PSDL_SYSTEM.EXCEPTION_ERROR;
           else
             BUFFER := (MODE          => NORMAL,
                        INITIALIZED => true,
                        NEW_DATA    => true,
                        N_VALUE     => VALUE);

           end if;
         end PUT;
       or
         accept PUT (VALUE : in STRING) do
         -- VSTRING.assign(VALUE,TempExcp);
           BUFFER := (MODE          => EXCPTION,
                      INITIALIZED => true,
                      NEW_DATA    => true,
                      E_VALUE     => TempExcp);

         end PUT;
       or
         accept IS_EXCEPTION (NAME : in PSDL_STRINGS.VSTRING;
                              CHECK : out BOOLEAN) do
           if BUFFER.MODE = EXCPTION then
             CHECK := VSTRING.equal(BUFFER.E_VALUE , NAME) ;
           else
             CHECK := false;
           end if;
         end IS_EXCEPTION;
       or
         accept IS_NORMAL (CHECK : out BOOLEAN) do
           CHECK := BUFFER.MODE = NORMAL;
         end IS_NORMAL;
       or
         terminate;
       end select;
     end loop;
   end DATA_STREAM;
 end SAMPLED_STREAM;

package body DATAFLOW_STREAM is

  type DATA_STREAM_TOKEN  is
    record
      INITIALIZED,
      NEW_DATA    : BOOLEAN := false;
      VALUE       : ELEMENT_TYPE;
    end record;
```

```
task body DATA_STREAM is

   BUFFER : DATA_STREAM_TOKEN;

begin
  loop
    select
      accept CHECK (NEW_DATA : out BOOLEAN) do
        NEW_DATA := BUFFER.NEW_DATA;
      end CHECK;

    or
      accept GET (OUTVALUE : out ELEMENT_TYPE) do
        if not (BUFFER.INITIALIZED and BUFFER.NEW_DATA) then
          raise PSDL_SYSTEM.BUFFER_UNDERFLOW;
        else
          OUTVALUE := BUFFER.VALUE;
          BUFFER.NEW_DATA := false;
        end if;
      end GET;

    or
      accept PUT (INVALUE : in ELEMENT_TYPE) do
        if BUFFER.NEW_DATA then
          raise PSDL_SYSTEM.BUFFER_OVERFLOW;
        else
          BUFFER.VALUE := INVALUE;
          BUFFER.NEW_DATA := true;
          BUFFER.INITIALIZED := true;
        end if;
      end PUT;

    or
      terminate;
    end select;
  end loop;
end DATA_STREAM;

function FRESH return BOOLEAN is
  RESULT : BOOLEAN;
begin
  DATA_STREAM.CHECK(RESULT);
  return(RESULT);
end FRESH;

end DATAFLOW_STREAM;
```

```
package body SAMPLED_STATE_VAR is
  type DATA_STREAM_TOKEN  is
    record
      INITIALIZED,
      NEW_DATA    : BOOLEAN := false;
      VALUE       : ELEMENT_TYPE := INITIAL_VALUE;
    end record;

  task body DATA_STREAM  is

    BUFFER : DATA_STREAM_TOKEN;

  begin
    loop
      select
        accept CHECK (NEW_DATA : out BOOLEAN) do
          NEW_DATA := BUFFER.NEW_DATA;
        end CHECK;

      or
        accept GET (OUTVALUE : out ELEMENT_TYPE) do
          if not BUFFER.INITIALIZED then
            raise PSDL_SYSTEM.BUFFER_UNDERFLOW;
          else
            OUTVALUE := BUFFER.VALUE;
            BUFFER.NEW_DATA := false;
          end if;
        end GET;

      or
        accept PUT (INVALUE : in ELEMENT_TYPE) do
          BUFFER.VALUE   :=  INVALUE;
          BUFFER. INITIALIZED := true;
          BUFFER.NEW_DATA      := true;
        end PUT;

      or
        terminate;
      end select;
    end loop;
  end DATA_STREAM;

  function FRESH return BOOLEAN is
    RESULT  :  BOOLEAN;
  begin
    DATA_STREAM.CHECK(RESULT);
    return RESULT;
  end FRESH;

end SAMPLED_STATE_VAR;
```

274

```ada
package body DATAFLOW_STATE_VAR is

  type DATA_STREAM_TOKEN is
    record
      INITIALIZED,
      NEW_DATA    : BOOLEAN := false;
      VALUE   :   ELEMENT_TYPE := INITIAL_VALUE;
    end record;

  task body DATA_STREAM is

    BUFFER : DATA_STREAM_TOKEN;
  begin
    loop
      select
        accept CHECK (NEW_DATA : out BOOLEAN) do
          NEW_DATA := BUFFER.NEW_DATA;
        end CHECK;

      or
        accept GET (OUTVALUE : out ELEMENT_TYPE) do
          if not (BUFFER.INITIALIZED and BUFFER.NEW_DATA) then
            raise PSDL_SYSTEM.BUFFER_UNDERFLOW;
          else
            OUTVALUE := BUFFER.VALUE;
            BUFFER.NEW_DATA := false;
          end if;
        end GET;

      or
        accept PUT (INVALUE : in ELEMENT_TYPE) do
          if BUFFER.NEW_DATA then
            raise PSDL_SYSTEM.BUFFER_OVERFLOW;
          else
            BUFFER.VALUE := INVALUE;
            BUFFER.NEW_DATA := true;
            BUFFER.INITIALIZED := true;
          end if;
        end PUT;

      or
        terminate;
      end select;
    end loop;
  end DATA_STREAM;
```

```
    function FRESH return BOOLEAN is
      RESULT  :  BOOLEAN;
    begin
      DATA_STREAM.CHECK(RESULT);
      return(RESULT);
    end FRESH;

  end DATAFLOW_STATE_VAR;

end PSDL_SYSTEM;
```

APPENDIX O KODIYAK SPECIFICATIONS FOR STATIC SCHEDULER

```
|-------------------------------------------------------------------
|
| file:      pre_ss.k
| author:    laura marlowe
| date:      dec 88
| modified:  dec 89 by laura j. white
|
|-------------------------------------------------------------------


|definitions of lexical classes

%define Digit    :[0-9]
%define Int      :[Digit]+
%define Letter   :[a-zA-Z_]
%define Alpha    :([Letter]|[Digit])
%define Blank    :[ \n]
%define Char     :[^()]
%define Quote    :["]


| definitions of white space

            :[Blank]+


| definitions of compound symbols and keywords

GTE              :">="
LTE              :"<="
NEQV             :"/="
ARROW            :"->"
TYPE             :type|TYPE
OPERATOR         :operator|OPERATOR
SPECIFICATION    :specification|SPECIFICATION
END              :end|END
GENERIC          :generic|GENERIC
INPUT            :input|INPUT
OUTPUT           :output|OUTPUT
STATES           :states|STATES
INITIALLY        :initially|INITIALLY
EXCEPTIONS       :exceptions|EXCEPTIONS
MAX_EXEC_TIME    :maximum[ ]execution[ ]time|MAXIMUM[ ]EXECUTION[ ]TIME
MAX_RESP_TIME    :maximum[ ]response[ ]time|MAXIMUM[ ]RESPONSE[ ]TIME
MIN_CALL_PERIOD  :minimum[ ]calling[ ]period|MINIMUM[ ]CALLING[ ]PERIOD
MICROSEC         :microsec|MICROSEC|us
```

277

```
MS                  :ms|MS
SEC                 :sec|SEC
MIN                 :min|MIN
HOURS               :hours|HOURS|hrs|HRS|hr|HR
BY                  :by( )requirements|BY( )REQUIREMENTS
KEYWORDS            :keywords|KEYWORDS
DESCRIPTION         :description|DESCRIPTION
AXIOMS              :axioms|AXIOMS
IMPLEMENTATION      :implementation|IMPLEMENTATION
ADA                 :ada|Ada|ADA
GRAPH               :graph|GRAPH
DATA_STREAM         :data( )stream|DATA( )STREAM
TIMER               :timer|TIMER
CONTROL             :control( )constraints|CONTROL( )CONSTRAINTS
TRIGGERED           :triggered|TRIGGERED
ALL                 :by( )all|BY( )ALL
SOME                :by( )some|BY( )SOME
PERIOD              :period|PERIOD
FINISH              :finish( )within|FINISH( )WITHIN
EXCEPTION           :exception|EXCEPTION
READ                :read( )timer|READ( )TIMER
RESET               :reset( )timer|RESET( )TIMER
START               :start( )timer|START( )TIMER
STOP                :stop( )timer|STOP( )TIMER
IF                  :if|IF
NOT                 :"~"|"not"|"NOT"
AND                 :"&"|"and"|"AND"
OR                  :"|"|"or"|"OR"
TRUE                :true|TRUE
FALSE               :false|FALSE
ID                  :(Letter)(Alpha)*
STRING_LITERAL      :(Quote)(Char)*(Quote)
INTEGER_LITERAL     :(Int)
REAL_LITERAL        :(Int)"."(Int)
TEXT                :"("(Char)*")"


| operator precedences
| %left means group and evaluate from the left

%left    OR;
%left    AND;
%left    NOT;
%left    '<', '>', '=', GTE, LTE, NEQV;
%left    ':';

%%
```

| attribute declarations for nonterminal symbols

```
start ( trn: string; );
psdl ( trn: string; );
component ( trn: string; );
data_type ( trn: string; );
operator ( trn: string; );
type_spec ( trn: string; );
type_decl_1_list ( trn: string; );
type_decl ( trn: string; );
op_spec_0_list ( trn: string; );
operator_spec ( trn: string; );
interface ( trn: string; );
attribute ( trn: string; );
time ( trn: string; );
unit ( value: int; );
id_list ( trn: string; );
reqmts_trace ( trn: string; );
functionality ( trn: string; );
keywords ( trn: string; );
informal_desc ( trn: string; );
formal_desc ( trn: string; );
type_impl ( trn: string; );
op_impl_0_list ( trn: string; );
operator_impl ( trn: string; children: string;);
psdl_impl ( trn: string; children: string; );
data_flow_diagram ( trn: string; );
link_0_list ( trn: string; );
link ( trn: string; );
opt_time ( trn: string; );
streams ( trn: string; );
type_name ( trn: string; );
timers ( trn: string; );
control_constraints ( trn: string; children: string; );
constraint_options ( trn: string; children: string; );
opt_trig ( trn: string; );
trigger ( trn: string; );
opt_per ( trn: string; );
opt_fin_w ( trn: string; );
timer_op ( trn: string; );
opt_if_predicate ( trn: string; );
predicate ( trn: string; );
expression_list ( trn: string; );
expression ( trn: string; );
relation (trn: string; );
simple_expression ( trn: string; );
exception_expr (trn: string; );
rel_op (trn: string; );
sign (trn: string; );
```

```
!attribute declarations for terminal symbols

ID( %text: string; );
TEXT( %text: string; );
STRING_LITERAL( %text: string; );
INTEGER_LITERAL( %text: string; );
REAL_LITERAL( %text: string; );


%%


!psdl grammar

start
    : psdl
     ( %output(psdl.trn); )
    ;

psdl
    : psdl component
      ( psdl[1].trn = [psdl[2].trn,component.trn]; )
    |
      ( psdl[1].trn = ""; )
    ;

component

    : data_type
      ( component.trn = data_type.trn; )
    | operator
      ( component.trn = operator.trn; )
    ;

data_type
    : TYPE ID type_spec type_impl
      ( data_type.trn = [type_spec.trn,type_impl.trn];  )
    ;

operator
    : OPERATOR  ID operator_spec operator_impl
      ( operator.trn = ["LINEAGE","\n",ID.%text,"\n",
                            operator_impl.children,"END LINEAGE","\n",
                            ID.%text,"\n",operator_spec.trn,
                            operator_impl.trn]; )
    ;

type_spec
    : SPECIFICATION type_decl_1_list op_spec_0_list functionality END
      ( type_spec.trn =  op_spec_0_list.trn; )
    ;
```

```
type_decl_1_list
    : type_decl
      ( type_decl_1_list.trn = type_decl.trn; )
    |
      (type_decl_1_list.trn = ""; )
    ;


type_decl
    : id_list ':' type_name
      ( type_decl.trn = id_list.trn; )
        | id_list ':' type_name ',' type_decl
          ( type_decl.trn = [id_list.trn,type_decl.trn]; )
    ;


op_spec_0_list
    : op_spec_0_list OPERATOR ID operator_spec
      ( op_spec_0_list[1].trn = [op_spec_0_list[2].trn,ID.&text,"\n",
                                 operator_spec.trn]; )
        |
      ( op_spec_0_list.trn = ""; )
    ;


operator_spec
    : SPECIFICATION interface functionality END
      ` operator_spec.trn = interface.trn; )


inter.
    : interface attribute reqmts_trace
      ( interface[1].trn = [interface[2].trn,attribute.trn]; )
    |
      (interface.trn = ""; )
    ;


attribute
    : GENERIC type_decl
      ( attribute.trn = ""; )
    | INPUT type_decl
      ( attribute.trn = ""; )
    | OUTPUT type_decl
      ( attribute.trn = ""; )
    | STATES type_decl INITIALLY expression_list
      ( attribute.trn = ["STATE","\n",type_decl.trn,"ENDSTATE","\n"]; )
    | EXCEPTIONS id_list
      ( attribute.trn = ""; )
    | MAX_EXEC_TIME time
      ( attribute.trn = ["MET","\n",time.trn,"\n"]; )
        | MIN_CALL_PERIOD time
      ( attribute.trn = ["MCP","\n",time.trn,"\n"]; )
        | MAX_RESP_TIME time
```

281

```
                ( attribute.trn = ["MRT","\n",time.trn,"\n"]; )
                ;

    id_list
        : ID ',' id_list
          ( id_list[1].trn =  [ID.%text,"\n",id_list[2].trn] ; )
        | ID
          ( id_list[1].trn =  [ID.%text,"\n"]; )
        ;


    time
        : INTEGER_LITERAL unit
          ( time.trn = i2s(s2i(INTEGER_LITERAL.%text)*unit.value); )
        ;


    unit
        : MICROSEC
          ( unit.value = 1; )
        | MS
          ( unit.value = 1000; )
        | SEC
          ( unit.value = 1000000; )
        | MIN
          ( unit.value = 60000000; )
        | HOURS
          ( unit.value = 3600000000; )
        |
          ( unit.value = 1000; )
        ;


    reqmts_trace
        : BY id_list
          ( reqmts_trace.trn = ""; )
        |
          ( reqmts_trace.trn = ""; )
          ;

    functionality
        : keywords informal_desc formal_desc
          ( functionality.trn = ""; )
        ;


    keywords
        : KEYWORDS id_list
          ( keywords.trn = "\n"; )
        |
          ( keywords.trn = ""; )
        ;
```

```
informal_desc
    : DESCRIPTION TEXT
      ( informal_desc.trn = "\n"; )
    |
      (informal_desc.trn = ""; )
    ;


formal_desc
    : AXIOMS TEXT
      ( formal_desc.trn = "\n"; )
    |
      ( formal_desc.trn = ""; )
    ;


type_impl
    : IMPLEMENTATION ADA ID END
      ( type_impl.trn = [ID.%text,"\n"]; )
    | IMPLEMENTATION type_name op_impl_0_list END
      ( type_impl.trn = op_impl_0_list.trn; )
    ;


op_impl_0_list
    : op_impl_0_list OPERATOR  ID operator_impl
      ( op_impl_0_list[1].trn = [op_impl_0_list[2].trn,ID.%text,"\n",
                                 operator_impl.trn]; )
    |
      ( op_impl_0_list[1].trn = ""; )
    ;


operator_impl
    : IMPLEMENTATION ADA ID END
      ( operator_impl.trn = [ID.%text,"\n"];
          operator_impl.children = ["ATOMIC","\n"]; )
    | IMPLEMENTATION psdl_impl
      ( operator_impl.trn = psdl_impl.trn;
          operator_impl.children = psdl_impl.children;)
    ;


psdl_impl
    : data_flow_diagram streams timers control_constraints informal_desc END
      ( psdl_impl.trn = [data_flow_diagram.trn,control_constraints.trn];
          psdl_impl.children = control_constraints.children;)


data_flow_diagram
    : GRAPH link_0_list
      ( data_flow_diagram.trn = link_0_list.trn; )
    ;
```

```
link_0_list
    : link_0_list link
      ( link_0_list[1].trn = [link_0_list[2].trn,link.trn]; )
    |
      ( link_0_list.trn = ""; )
    ;


link
    : ID '.' ID opt_time ARROW ID
      ( link.trn = ["LINK","\n",ID[1].%text,"\n",ID[2].%text,"\n",
                    opt_time.trn,"\n",ID[3].%text,"\n"]; )
    ;


opt_time
    : ':' time
      ( opt_time.trn = time.trn; )
    |
      ( opt_time.trn = "0"; )
    ;



streams
    : DATA_STREAM type_decl
      ( streams.trn = ""; )
    |
      (streams.trn = ""; )
    ;


type_name
    : ID '[' type_decl ']'
      ( type_name.trn = "" ; )
    | ID
      ( type_name.trn = "" ; )
    ;


timers
    : TIMER id_list
      ( timers.trn = "" ; )
    |
      (timers.trn = ""; )
        ;


control_constraints
    : CONTROL
      ( control_constraints.trn = "";
            control_constraints.children ="";)
    | CONTROL OPERATOR ID opt_trig opt_per opt_fin_w constraint_options
      (control_constraints.trn = [ID.%text,"\n",opt_per.trn,opt_fin_w.trn,
                                   constraint_options.trn];
            control_constraints.children = [ID.%text, "\n",
                                            constraint_options.children]; )
```

284

```
                     |
                        (control_constraints.trn = "";
                         control_constraints.children = "";)
            ;


constraint_options
            :  OUTPUT id_list IF predicate reqmts_trace constraint_options
               ( constraint_options[1].trn = constraint_options[2].trn;
                 constraint_options.children = constraint_options[2].children;)
            | EXCEPTION ID opt_if_predicate reqmts_trace constraint_options
               ( constraint_options[1].trn = constraint_options[2].trn;
                 constraint_options.children = constraint_options[2].children;)
            | timer_op ID  opt_if_predicate reqmts_trace constraint_options
               ( constraint_options[1].trn = constraint_options[2].trn;
                 constraint_options.children = constraint_options[2].children;)
            | OPERATOR ID opt_trig opt_per opt_fin_w constraint_options
               ( constraint_options[1].trn = [ID.$text,"\n",opt_per.trn,
                             opt_fin_w.trn,constraint_options[2].trn];
                 constraint_options.children = [ID.$text, "\n",
                                      constraint_options[2].children];)
            |
               ( constraint_options.trn = "";
                 constraint_options.children = "";)
            ;


opt_trig
        : TRIGGERED trigger opt_if_predicate reqmts_trace
          ( opt_trig.trn = ""; )
        |
          (opt_trig.trn = ""; )
        ;


trigger
        : ALL id_list
          ( trigger.trn = ""; )
        | SOME id_list
          ( trigger.trn = ""; )
        |
          (trigger.trn = ""; )
        ;


opt_per
        :PERIOD time reqmts_trace
         ( opt_per.trn = ["PERIOD","\n",time.trn,"\n"]; )
        |
         ( opt_per.trn = ""; )
        ;


opt_fin_w
        : FINISH time reqmts_trace
          ( opt_fin_w.trn = ["WITHIN","\n",time.trn,"\n"]; )
```

```
       |
        ( opt_fin_w.trn = ""; )
       ;



timer_op
    : READ
         ( timer_op.trn = ""; )
       | RESET
      ( timer_op.trn = ""; )
    | START
      ( timer_op.trn = ""; )
    | STOP
      ( timer_op.trn = ""; )
    ;


opt_if_predicate
    : IF predicate
      ( opt_if_predicate.trn = ""; )
        |
          (opt_if_predicate.trn = ""; )
    ;



expression_list
    : expression
      (expression_list.trn = ""; )
       | expression ',' expression_list
      (expression_list(1).trn = ""; )
    ;

expression
        : INTEGER_LITERAL
          (expression.trn = ""; )
        | REAL_LITERAL
          (expression.trn = ""; )
        | STRING_LITERAL
          (expression.trn = ""; )
        | TRUE
          (expression.trn = ""; )
        | FALSE
          (expression.trn = ""; )
        | ID
          (expression.trn = ""; )
        | type_name '.' ID '(' expression_list ')'
          (expression.trn = ""; )
        ;
```

```
predicate
        : relation
          (predicate.trn = ""; )
        | relation AND predicate
          (predicate[1].trn = ""; )
        | relation OR predicate
          (predicate[1].trn = ""; )
        ;


relation
        : simple_expression rel_op simple_expression
          (relation.trn = ""; )
        | simple_expression
          (relation.trn = ""; )
        ;

simple_expression
        : sign INTEGER_LITERAL unit
          (simple_expression.trn = ""; )
        | sign REAL_LITERAL
          (simple_expression.trn = ""; )
        | ID
          (simple_expression.trn = ""; )
        | STRING_LITERAL
          (simple_expression.trn = ""; )
        | '(' predicate ')'
          (simple_expression.trn = ""; )
        | NOT ID
          (simple_expression.trn = "";)
        | NOT '(' predicate ')'
          (simple_expression.trn = ""; )
        | TRUE
          (simple_expression.trn = ""; )
        | FALSE
          (simple_expression.trn = ""; )
        | NOT TRUE
          (simple_expression.trn = ""; )
        | NOT FALSE
          (simple_expression.trn = ""; )
        ;

rel_op
      : '<'
        (rel_op.trn = ""; )
        | '>'
        (rel_op.trn = ""; )
        | '='
        (rel_op.trn = ""; )
        | GTE
        (rel_op.trn = ""; )
```

287

```
        | LTE
    {rel_op.trn = ""; }
        | NEQV
    {rel_op.trn = ""; }
        | ':'
        {rel_op.trn = ""; }
        ;

sign
        : '+'
        {sign  rn = ""; }
        | '-'
        {sigi    'n = ""; }
        |
        {sign.trn = ""; }
        ;
```

APPENDIX P STATIC SCHEDULER DRIVER

```
-------------------------------------------------------------------
-- file:     driver.a
-- author:   murat kilic
-- date:     nov 89
-- modified: 26 dec 89 by laura j. white
-------------------------------------------------------------------


with TEXT_IO;
with FILES; use FILES;
with FILE_PROCESSOR;
with EXCEPTION_HANDLER;
with TOPOLOGICAL_SORTER;
with HARMONIC_BLOCK_BUILDER;
with OPERATOR_SCHEDULER;

procedure STATIC_SCHEDULER is

   THE_GRAPH        : DIGRAPH.GRAPH;
   PRECEDENCE_LIST  : DIGRAPH.V_LISTS.LIST;
   SCH_INPUTS       : SCHEDULE_INPUTS_LIST.LIST;
   AGENDA           : SCHEDULE_INPUTS_LIST.LIST;
   BASE_BLOCK       : INTEGER;
   H_B_LENGTH       : INTEGER;
   STOP_TIME        : INTEGER := 0;

begin
   FILE_PROCESSOR.SEPARATE_DATA (THE_GRAPH);
   FILE_PROCESSOR.VALIDATE_DATA(THE_GRAPH);
   TOPOLOGICAL_SORTER.TOPOLOGICAL_SORT(THE_GRAPH, PRECEDENCE_LIST);
   HARMONIC_BLOCK_BUILDER.CALC_PERIODIC_EQUIVALENTS (PRECEDENCE_LIST);
   HARMONIC_BLOCK_BUILDER.FIND_BASE_BLOCK (PRECEDENCE_LIST, BASE_BLOCK);
   HARMONIC_BLOCK_BUILDER.FIND_BLOCK_LENGTH (PRECEDENCE_LIST,H_B_LENGTH);
   OPERATOR_SCHEDULER.TEST_DATA(PRECEDENCE_LIST, H_B_LENGTH);
   loop
      if NOT(TEST_VERIFIED) then
         TEXT_IO.PUT("Although a schedule may be possible, there is no ");
         TEXT_IO.PUT_LINE("guarantee that it will execute");
         TEXT_IO.PUT_LINE("within the required timing constraints.");
         TEXT_IO.NEW_LINE;
      end if;
      begin
         OPERATOR_SCHEDULER.SCHEDULE_INITIAL_SET
                  (PRECEDENCE_LIST,SCH_INPUTS,H_B_LENGTH,STOP_TIME);
```

```
      OPERATOR_SCHEDULER.SCHEDULE_REST_OF_BLOCK
               (PRECEDENCE_LIST,SCH_INPUTS,H_B_LENGTH,STOP_TIME);
      OPERATOR_SCHEDULER.CREATE_STATIC_SCHEDULE
               (THE_GRAPH,SCH_INPUTS,H_B_LENGTH);
      TEXT_IO.PUT("A feasible schedule found, ");
      TEXT_IO.PUT_LINE("the Harmonic Block with Precedence ");
      TEXT_IO.PUT_LINE(" Constraints Scheduling Algorithm Used. ");
      SCH_INPUTS := null;
      exit;
   exception
      when OPERATOR_SCHEDULER.MISSED_DEADLINE =>
         null;
      when OPERATOR_SCHEDULER.OVER_TIME =>
         null;
   end;

   begin
      HARMONIC_BLOCK_BUILDER.CALC_PERIODIC_EQUIVALENTS
                  (THE_GRAPH.VERTICES);
      OPERATOR_SCHEDULER.SCHEDULE_WITH_EARLIEST_START
               (THE_GRAPH,AGENDA,H_B_LENGTH);
      OPERATOR_SCHEDULER.CREATE_STATIC_SCHEDULE
               (THE_GRAPH,AGENDA,H_B_LENGTH);
      TEXT_IO.PUT_LINE("A feasible schedule found, the Earliest Start");
      TEXT_IO.PUT_LINE("Scheduling Algorithm Used. ");
      AGENDA := null;
      exit;
   exception
      when OPERATOR_SCHEDULER.MISSED_DEADLINE =>
         null;
      when OPERATOR_SCHEDULER.OVER_TIME =>
         null;
   end;

   begin
      OPERATOR_SCHEDULER.SCHEDULE_WITH_EARLIEST_DEADLINE
               (THE_GRAPH,AGENDA,H_B_LENGTH);
      OPERATOR_SCHEDULER.CREATE_STATIC_SCHEDULE
               (THE_GRAPH,AGENDA,H_B_LENGTH);
      TEXT_IO.PUT_LINE("A feasible schedule found, the Earliest ");
      TEXT_IO.PUT_LINE("Deadline Scheduling Algorithm Used. ");
      AGENDA := null;
      exit;
   exception
      when OPERATOR_SCHEDULER.MISSED_DEADLINE =>
         null;
      when OPERATOR_SCHEDULER.OVER_TIME =>
         null;
      when OPERATOR_SCHEDULER.MISSED_OPERATOR =>
         null;
   end;
```

```
    end loop;

exception
  when FILE_PROCESSOR.CRIT_OP_LACKS_MET =>
    EXCEPTION_HANDLER.CRIT_O_L_MET(Exception_Operator);

  when FILE_PROCESSOR.MET_NOT_LESS_THAN_PERIOD =>
    EXCEPTION_HANDLER.MET_N_L_T_PERIOD(Exception_Operator);

  when FILE_PROCESSOR.MET_NOT_LESS_THAN_MRT =>
    EXCEPTION_HANDLER.MET_N_L_T_MRT(Exception_Operator);

  when FILE_PROCESSOR.MCP_NOT_LESS_THAN_MRT =>
    EXCEPTION_HANDLER.MCP_N_L_T_MRT(Exception_Operator);

  when FILE_PROCESSOR.MCP_LESS_THAN_MET =>
    EXCEPTION_HANDLER.MCP_L_T_MET(Exception_Operator);

  when FILE_PROCESSOR.MET_IS_GREATER_THAN_FINISH_WITHIN =>
    EXCEPTION_HANDLER.MET_I_G_T_FINISH_WITHIN(Exception_Operator);

  when FILE_PROCESSOR.SPORADIC_OP_LACKS_MCP =>
    EXCEPTION_HANDLER.SPORADIC_O_L_MCP(Exception_Operator);

  when FILE_PROCESSOR.SPORADIC_OP_LACKS_MRT =>
    EXCEPTION_HANDLER.SPORADIC_O_L_MRT(Exception_Operator);

  when SCHEDULE_INPUTS_LIST.BAD_VALUE =>
    EXCEPTION_HANDLER.S_I_L_BAD_VALUE;

  when DIGRAPH.V_LISTS.BAD_VALUE =>
    EXCEPTION_HANDLER.V_L_BAD_VALUE;

  when DIGRAPH.E_LISTS.BAD_VALUE =>
    EXCEPTION_HANDLER.E_L_BAD_VALUE;

  when HARMONIC_BLOCK_BUILDER.NO_BASE_BLOCK =>
    EXCEPTION_HANDLER.NO_B_BLOCK;

  when HARMONIC_BLOCK_BUILDER.NO_OPERATOR_IN_LIST =>
    EXCEPTION_HANDLER.NO_OP_IN_LIST;

  when HARMONIC_BLOCK_BUILDER.MET_NOT_LESS_THAN_PERIOD =>
    EXCEPTION_HANDLER.MET_N_L_T_PERIOD(Exception_Operator);

end STATIC_SCHEDULER;
```

APPENDIX Q STATIC SCHEDULER ERROR HANDLER

```
----------------------------------------------------------------
-- file:       e_handler_s.a
-- author:     murat kilic
-- date:       nov 89
-- modified:   dec 89 murat kilic
----------------------------------------------------------------

with FILES; use FILES;

package EXCEPTION_HANDLER is

  procedure CRIT_O_L_MET(Exception_Operator : in OPERATOR_ID);

  procedure MET_N_L_T_PERIOD(Exception_Operator : in OPERATOR_ID);

  procedure MET_N_L_T_MRT(Exception_Operator : in OPERATOR_ID);

  procedure MCP_N_L_T_MRT(Exception_Operator : in OPERATOR_ID);

  procedure MCP_L_T_MET(Exception_Operator : in OPERATOR_ID);

  procedure MET_I_G_T_FINISH_WITHIN(Exception_Operator : in OPERATOR_ID);

  procedure PERIOD_L_T_FINISH_WITHIN(Exception_Operator : in OPERATOR_ID);

  procedure SPORADIC_O_L_MCP(Exception_Operator : in OPERATOR_ID);

  procedure SPORADIC_O_L_MRT(Exception_Operator : in OPERATOR_ID);

  procedure S_I_L_BAD_VALUE;

  procedure V_L_BAD_VALUE;

  procedure E_L_BAD_VALUE;

  procedure NO_B_BLOCK;

  procedure NO_OP_IN_LIST;

end EXCEPTION_HANDLER;
```

APPENDIX R STATIC SCHEDULER ERROR HANDLER

```
-----------------------------------------------------------------
-- file:     e_handler_b.a
-- author:   murat kilic
-- date:     nov 89
-- modified: dec 89 by murat kilic
-----------------------------------------------------------------


with TEXT_IO;
with FILES; use FILES;

package body EXCEPTION_HANDLER is

  procedure CRIT_O_L_MET(Exception_Operator : in OPERATOR_ID) is
  begin
    TEXT_IO.PUT ("Critical Operator ");
    VARSTRING.PUT (Exception_Operator);
    TEXT_IO.PUT_LINE (" must have an MET");
  end CRIT_O_L_MET;

  procedure MET_N_L_T_PERIOD(Exception_Operator : in OPERATOR_ID) is
  begin
    TEXT_IO.PUT ("MET is greater than PERIOD in operator ");
    VARSTRING.PUT_LINE (Exception_Operator);
  end MET_N_L_T_PERIOD;

  procedure MET_N_L_T_MRT(Exception_Operator : in OPERATOR_ID) is
  begin
    TEXT_IO.PUT ("MET is greater than MRT in operator ");
    VARSTRING.PUT_LINE (Exception_Operator);
  end MET_N_L_T_MRT;

  procedure MCP_N_L_T_MRT(Exception_Operator : in OPERATOR_ID) is
  begin
    TEXT_IO.PUT ("MCP is greater than MRT in operator ");
    VARSTRING.PUT_LINE (Exception_Operator);
  end MCP_N_L_T_MRT;

  procedure MCP_L_T_MET(Exception_Operator : in OPERATOR_ID) is
  begin
    TEXT_IO.PUT ("MCP is less than MET in operator ");
    VARSTRING.PUT_LINE (Exception_Operator);
  end MCP_L_T_MET;
```

```
procedure MET_I_G_T_FINISH_WITHIN
          (Exception_Operator : in OPERATOR_ID) is
begin
  TEXT_IO.PUT ("MET is greater than FINISH_WITHIN in operator ");
  VARSTRING.PUT_LINE (Exception_Operator);
end MET_I_G_T_FINISH_WITHIN;

procedure PERIOD_L_T_FINISH_WITHIN
          (Exception_Operator : in OPERATOR_ID) is
begin
  TEXT_IO.PUT ("Period is less than FINISH_WITHIN in operator ");
  VARSTRING.PUT_LINE (Exception_Operator);
end PERIOD_L_T_FINISH_WITHIN;

procedure SPORADIC_O_L_MCP(Exception_Operator : in OPERATOR_ID) is
begin
  TEXT_IO.PUT ("Sporadic Operator ");
  VARSTRING.PUT (Exception_Operator);
  TEXT_IO.PUT_LINE (" must have an MCP");
end SPORADIC_O_L_MCP;

procedure SPORADIC_O_L_MRT(Exception_Operator : in OPERATOR_ID) is
begin
  TEXT_IO.PUT ("Sporadic Operator ");
  VARSTRING.PUT (Exception_Operator);
  TEXT_IO.PUT_LINE (" must have an MRT");
end SPORADIC_O_L_MRT;

procedure S_I_L_BAD_VALUE is
begin
  TEXT_IO.PUT ("You try to get a schedule input where your pointer ");
  TEXT_IO.PUT_LINE ("is pointing a null record.");
end S_I_L_BAD_VALUE;

procedure V_L_BAD_VALUE is
begin
  TEXT_IO.PUT ("You try to get an operator where your pointer ");
  TEXT_IO.PUT_LINE ("is pointing a null record.");
end V_L_BAD_VALUE;

procedure E_L_BAD_VALUE is
begin
  TEXT_IO.PUT ("You try to get a link data where your pointer ");
  TEXT_IO.PUT_LINE ("is pointing a null record.");
end E_L_BAD_VALUE;

procedure NO_B_BLOCK is
begin
  TEXT_IO.PUT_LINE ("There is no BASE BLOCK in this system.");
end NO_B_BLOCK;
```

```
procedure NO_OP_IN_LIST is
 begin
   TEXT_IO.PUT_LINE ("There is no CRITICAL OPERATOR in this system "
 end NO_OP_IN_LIST;

end EXCEPTION_HANDLER;
```

APPENDIX S STATIC SCHEDULER GLOBALS

```
------------------------------------------------------------------
-- file:       files.a
-- author:     murat kilic
-- date:       oct 89
-- modified:   dec 89 by laura j. white
------------------------------------------------------------------


with VSTRINGS;
with SEQUENCES;
with GRAPHS;

package FILES is

  package VARSTRING is new VSTRINGS(80);
  use VARSTRING;

  subtype OPERATOR_ID is VSTRING;
  subtype VALUE is NATURAL;
  subtype MET is VALUE;
  subtype MRT is VALUE;
  subtype MCP is VALUE;
  subtype PERIOD is VALUE;
  subtype WITHIN is VALUE;
  subtype STARTS is VALUE;
  subtype STOPS is VALUE;
  subtype LOWERS is VALUE;
  subtype UPPERS is VALUE;

  Exception_Operator : OPERATOR_ID;

  TEST_VERIFIED : BOOLEAN := TRUE;

  type OPERATOR is
    record
      THE_OPERATOR_ID   : OPERATOR_ID;
      THE_MET           : MET      := 0;
      THE_MRT           : MRT      := 0;
      THE_MCP           : MCP      := 0;
      THE_PERIOD        : PERIOD   := 0;
      THE_WITHIN        : WITHIN   := 0;
    end record;

  package DIGRAPH is new GRAPHS(OPERATOR);
```

```
   type SCHEDULE_INPUTS is
      record
        THE_OPERATOR     : OPERATOR_ID;
        THE_START        : STARTS   := 0;
        THE_STOP         : STOPS    := 0;
        THE_LOWER        : LOWERS   := 0;
        THE_UPPER        : UPPERS   := 0;
      end record;

   package SCHEDULE_INPUTS_LIST is new SEQUENCES(SCHEDULE_INPUTS);

   type OP_INFO is
      record
        NODE           : OPERATOR;
        SUCCESSORS     : DIGRAPH.V_LISTS.LIST;
        PREDICESSORS   : DIGRAPH.V_LISTS.LIST;
      end record;

   package OP_INFO_LIST is new SEQUENCES(OP_INFO);

end FILES;
```

APPENDIX T STATIC SCHEDULER FILE PROCESSOR

```
----------------------------------------------------------------
-- file:      fp_s.a
-- authors:   laura marlowe
--            murat kilic
-- date:      nov 89
----------------------------------------------------------------


with FILES; use FILES;
package FILE_PROCESSOR is

  procedure SEPARATE_DATA (THE_GRAPH : in out DIGRAPH.GRAPH);

  procedure VALIDATE_DATA (THE_GRAPH : in out DIGRAPH.GRAPH);

  CRIT_OP_LACKS_MET                    : exception;
  MET_NOT_LESS_THAN_PERIOD             : exception;
  MET_NOT_LESS_THAN_MRT                : exception;
  MCP_NOT_LESS_THAN_MRT                : exception;
  MCP_LESS_THAN_MET                    : exception;
  MET_IS_GREATER_THAN_FINISH_WITHIN    : exception;
  SPORADIC_OP_LACKS_MCP                : exception;
  SPORADIC_OP_LACKS_MRT                : exception;
  PERIOD_LESS_THAN_FINISH_WITHIN       : exception;

end FILE_PROCESSOR;
```

APPENDIX U STATIC SCHEDULER FILE PROCESSOR

```
-------------------------------------------------------------
-- file:      fp_b.a
-- author:    laura marlowe
--            murat kilic
-- date:      oct 89
-- modified:  dec 89 by laura j. white
-------------------------------------------------------------


with TEXT_IO;
with FILES; use FILES;

package body FILE_PROCESSOR  is

  procedure SEPARATE_DATA (THE_GRAPH : in out DIGRAPH.GRAPH) is

    -- This procedure reads the output file which has the link
    -- information with the Atomic operators and depending upon
    -- the keywords that are declared as constants separates the
    -- information in the file and stores it in the graph data
    -- structure, where GRAPH has the operator and link information
    -- in it.

    package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);

    MET     : constant VARSTRING.VSTRING := VARSTRING.VSTR("MET");
    MRT     : constant VARSTRING.VSTRING := VARSTRING.VSTR("MRT");
    MCP     : constant VARSTRING.VSTRING := VARSTRING.VSTR("MCP");
    PERIOD  : constant VARSTRING.VSTRING := VARSTRING.VSTR("PERIOD");
    WITHIN  : constant VARSTRING.VSTRING := VARSTRING.VSTR("WITHIN");
    LINK    : constant VARSTRING.VSTRING := VARSTRING.VSTR("LINK");
    ATOMIC  : constant VARSTRING.VSTRING := VARSTRING.VSTR("ATOMIC");
    EMPTY   : constant VARSTRING.VSTRING := VARSTRING.VSTR("EMPTY");

    Current_Value    : VALUE;
    New_Stream       : DIGRAPH.DATA_STREAM;
    New_Word         : VARSTRING.VSTRING;
    Cur_Opt          : OPERATOR;
    Cur_Link         : DIGRAPH.LINK_DATA;

    NON_CRITS  : TEXT_IO.FILE_TYPE;
    AG_OUTFILE : TEXT_IO.FILE_TYPE;
    INPUT      : TEXT_IO.FILE_MODE := TEXT_IO.IN_FILE;
    OUTPUT     : TEXT_IO.FILE_MODE := TEXT_IO.OUT_FILE;
```

```
PRINT_EDGES  : DIGRAPH.E_LISTS.LIST;
S1, S2, L1   : DIGRAPH.V_LISTS.LIST;
ID1, ID2     : OPERATOR;
START_NODE   : OPERATOR;
END_NODE     : OPERATOR;

procedure INITIALIZE_OPERATOR (OP : in out OPERATOR) is
begin
  OP.THE_MET    := 0;
  OP.THE_MRT    := 0;
  OP.THE_MCP    := 0;
  OP.THE_PERIOD := 0;
  OP.THE_WITHIN := 0;
end;

begin
  TEXT_IO.OPEN (AG_OUTFILE, INPUT, "/n/suns2/work/caps/prototypes/atomic.info");
  TEXT_IO.CREATE(NON_CRITS, OUTPUT, "/n/suns2/work/caps/prototypes/non_crits");
  VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
  while not TEXT_IO.END_OF_FILE(AG_OUTFILE) loop

    if VARSTRING.EQUAL (New_Word, LINK) then
      START_NODE.THE_OPERATOR_ID := EMPTY;
      END_NODE.THE_OPERATOR_ID := EMPTY;
      DIGRAPH.V_STRING.GET_LINE(AG_OUTFILE, New_Stream);
      Cur_Link.THE_DATA_STREAM := New_Stream;
      VARSTRING.GET_LINE(AG_OUTFILE, New_Word);
      L1 := THE_GRAPH.VERTICES;
      while DIGRAPH.V_LISTS.NON_EMPTY(L1) loop
        if VARSTRING.EQUAL(DIGRAPH.V_LISTS.VALUE(L1).THE_OPERATOR_ID,New_Word)
                                                                    then
          START_NODE := DIGRAPH.V_LISTS.VALUE(L1);
          exit;
        end if;
        DIGRAPH.V_LISTS.NEXT(L1);
      end loop;
      VALUE_IO.GET(AG_OUTFILE, Current_Value);
      TEXT_IO.SKIP_LINE(AG_OUTFILE);
      Cur_Link.THE_LINK_MET := Current_Value;
      VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
      L1 := THE_GRAPH.VERTICES;
      while DIGRAPH.V_LISTS.NON_EMPTY(L1) loop
        if VARSTRING.EQUAL(DIGRAPH.V_LISTS.VALUE(L1).THE_OPERATOR_ID,New_Word)
                                                                    then
          END_NODE := DIGRAPH.V_LISTS.VALUE(L1);
          exit;
        end if;
        DIGRAPH.V_LISTS.NEXT(L1);
      end loop;

      -- when either starting node or ending node of a link is
```

```
-- EXTERNAL, the link information will not be added to the
-- graph.  Assuming that all external data coming in is ready
-- at start time.                              .

if VARSTRING.NOTEQUAL(START_NODE.THE_OPERATOR_ID,EMPTY) and
            VARSTRING.NOTEQUAL(END_NODE.THE_OPERATOR_ID,EMPTY) then
    DIGRAPH.V_LISTS.ADD(START_NODE, Cur_Link.THE_FIRST_OP_ID);
    DIGRAPH.V_LISTS.ADD(END_NODE, Cur_Link.THE_SECOND_OP_ID);
    DIGRAPH.ADD(Cur_Link, THE_GRAPH);
end if;
VARSTRING.GET_LINE ( AG_OUTFILE, New_Word);

elsif VARSTRING.EQUAL (New_Word,ATOMIC) then
    VARSTRING.GET_LINE ( AG_OUTFILE, New_Word);
    Cur_Opt.THE_OPERATOR_ID := New_Word;
    VARSTRING.GET_LINE (AG_OUTFILE, New_Word);
    if (VARSTRING.EQUAL(New_Word, ATOMIC)) or
        (VARSTRING.EQUAL(New_Word, LINK)) or
        (TEXT_IO.END_OF_FILE(AG_OUTFILE)) then
            VARSTRING.PUT_LINE(NON_CRITS, Cur_Opt.THE_OPERATOR_ID);
    else
        while VARSTRING.NOTEQUAL (New_Word, ATOMIC) and
              VARSTRING.NOTEQUAL (New_Word, LINK)    and
              not TEXT_IO.END_OF_FILE(AG_OUTFILE) loop

        if VARSTRING.EQUAL (New_Word,MET) then
            VALUE_IO.GET(AG_OUTFILE,Current_Value);
            TEXT_IO.SKIP_LINE(AG_OUTFILE);
            Cur_Opt.THE_MET := Current_Value;

          elsif VARSTRING.EQUAL (New_Word,MRT) then
            VALUE_IO.GET(AG_OUTFILE,Current_Value);
            TEXT_IO.SKIP_LINE(AG_OUTFILE);
            Cur_Opt.THE_MRT:= Current_Value;

        elsif VARSTRING.EQUAL (New_Word,MCP) then
            VALUE_IO.GET(AG_OUTFILE,Current_Value);
            TEXT_IO.SKIP_LINE(AG_OUTFILE);
          Cur_Opt.THE_MCP := Current_Value;

          elsif VARSTRING.EQUAL (New_Word,PERIOD) then
            VALUE_IO.GET(AG_OUTFILE,Current_Value);
            TEXT_IO.SKIP_LINE(AG_OUTFILE);
            Cur_Opt.THE_PERIOD := Current_Value;

        elsif VARSTRING.EQUAL (New_Word,WITHIN) then
            VALUE_IO.GET(AG_OUTFILE,Current_Value);
            TEXT_IO.SKIP_LINE(AG_OUTFILE);
            Cur_Opt.THE_WITHIN := Current_Value;
        end if;
```

```
            VARSTRING.GET_LINE(AG_OUTFILE,New_Word);
        end loop;

        DIGRAPH.ADD(Cur_Opt, THE_GRAPH);
          INITIALIZE_OPERATOR(Cur_Opt);
        end if;
      end if;
    end loop;
end SEPARATE_DATA;



procedure VALIDATE_DATA (THE_GRAPH : in out DIGRAPH.GRAPH) is

  TARGET : DIGRAPH.V_LISTS.LIST;
  package VAL_IO is new TEXT_IO.INTEGER_IO(VALUE);
begin
  TARGET := THE_GRAPH.VERTICES;
  while DIGRAPH.V_LISTS.NON_EMPTY(TARGET) loop

    -- ensure that there is no operator without an MET.
    if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MET = 0 then
      Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
      raise CRIT_OP_LACKS_MET;
    end if;

    if DIGRAPH.V_LISTS.VALUE(TARGET).THE_PERIOD = 0 then
      -- Check to ensure that MCP has a value for sporadic operators
      if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MCP = 0 then
        Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
        raise SPORADIC_OP_LACKS_MCP;
      elsif  DIGRAPH.V_LISTS.VALUE(TARGET).THE_MET >
                                      DIGRAPH.V_LISTS.VALUE(TARGET).THE_MCP then
        Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
        raise MCP_LESS_THAN_MET;
      end if;

      -- Check to ensure that MRT has a value for sporadic operators
      if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MRT = 0 then
        Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
        raise SPORADIC_OP_LACKS_MRT;
      end if;

      -- Check to ensure that the MRT is greater than the MET.
      if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MET >
                                      DIGRAPH.V_LISTS.VALUE(TARGET).THE_MRT then
        Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
        raise MET_NOT_LESS_THAN_MRT;
      end if;

      -- Guarantees that an operator can fire at least once
```

```
               -- before a response expected.
               if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MCP >
                                         DIGRAPH.V_LISTS.VALUE(TARGET).THE_MRT then
                 raise MCP_NOT_LESS_THAN_MRT;
               end if;

           else
               -- Check to ensure that the PERIOD is greater than the MET.
               if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MET >
                                         DIGRAPH.V_LISTS.VALUE(TARGET).THE_PERIOD then
                 Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
                 raise MET_NOT_LESS_THAN_PERIOD;
               end if;

               -- Check to ensure that the FINISH_WITHIN is greater than the MET.
               if DIGRAPH.V_LISTS.VALUE(TARGET).THE_WITHIN /= 0 then
                 if DIGRAPH.V_LISTS.VALUE(TARGET).THE_MET >
                                         DIGRAPH.V_LISTS.VALUE(TARGET).THE_WITHIN then
                   Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
                   raise MET_IS_GREATER_THAN_FINISH_WITHIN;
                 elsif DIGRAPH.V_LISTS.VALUE(TARGET).THE_PERIOD <
                       DIGRAPH.V_LISTS.VALUE(TARGET).THE_WITHIN then
                   Exception_Operator := DIGRAPH.V_LISTS.VALUE(TARGET).THE_OPERATOR_ID;
                   raise PERIOD_LESS_THAN_FINISH_WITHIN;
                 end if;
               end if;

           end if;

           DIGRAPH.V_LISTS.NEXT(TARGET);
         end loop;
       end VALIDATE_DATA;

   end FILE_PROCESSOR;
```

APPENDIX V STATIC SCHEDULER GRAPH STRUCTURE

```ada
--------------------------------------------------------------
-- file:      graphs_s.a
-- authors    murat kilic
--            isaac mostov
--            tony davis
-- date:      sep 89
-- modified:  oct 89 by murat kilic
--------------------------------------------------------------


with SEQUENCES;
with VSTRINGS;

generic
  type VERTEX is private;

package GRAPHS is

  package V_LISTS is new SEQUENCES(VERTEX);
  use V_LISTS;

  package V_STRING is new VSTRINGS(80);
  use V_STRING;

  subtype DATA_STREAM is VSTRING;
  subtype MET is NATURAL;

  type LINK_DATA is
    record
      THE_DATA_STREAM   : DATA_STREAM:
      THE_FIRST_OP_ID   : V_LISTS.LIST;
      THE_LINK_MET      : MET := 0;
      THE_SECOND_OP_ID  : V_LISTS.LIST;
    end record;

  package E_LISTS is new SEQUENCES(LINK_DATA);
  use E_LISTS;

  type GRAPH is
    record
      VERTICES : V_LISTS.LIST;
      LINKS    : E_LISTS.LIST;
    end record;
```

304

```
function EQUAL_GRAPHS(G1 : in GRAPH; G2 : in GRAPH) return BOOLEAN;

procedure EMPTY(G : out GRAPH);

function IS_NODE(X : in VERTEX; G : GRAPH) return BOOLEAN;

function IS_LINK(X : in VERTEX; Y : in VERTEX;
                                G : in GRAPH) return BOOLEAN;

procedure ADD(X : in VERTEX; G : in out GRAPH);

procedure ADD(L : in LINK_DATA; G : in out GRAPH);

procedure REMOVE(X : in VERTEX; G : in out GRAPH);

procedure REMOVE(X : in VERTEX; Y : in VERTEX; G : in out GRAPH);

procedure SCAN_NODES(G : in GRAPH; S : in out V_LISTS.LIST);

procedure SCAN_PARENTS(X : in VERTEX; G : in GRAPH;
                                      S : in out V_LISTS.LIST);

procedure SCAN_CHILDREN(X : in VERTEX; G : in GRAPH;
                                       S : in out V_LISTS.LIST);

procedure DUPLICATE(G1 : in GRAPH; G2 : in out GRAPH);

procedure T_SORT(G : in GRAPH; S : in out V_LISTS.LIST);

end GRAPHS;
```

APPENDIX W STATIC SCHEDULER GRAPH STRUCTURE

```
---------------------------------------------------------------------
-- file:       graphs_b.a
-- author:     murat kilic
--             isaac mostov
--             tony davis
-- date:       sep 89
-- modified:   oct 89 by murat kilic
---------------------------------------------------------------------


with UNCHECKED_DEALLOCATION;

package body GRAPHS is

  procedure FREE is new UNCHECKED_DEALLOCATION(E_LISTS.NODE, E_LISTS.LIST);

  function EQUAL_GRAPHS(G1 : in GRAPH; G2 : in GRAPH) return BOOLEAN is

    function SUB_SET(G1 : in GRAPH; G2 : in GRAPH) return BOOLEAN is
      L1 : V_LISTS.LIST := G1.VERTICES;
      L2 : E_LISTS.LIST := G1.LINKS;
    begin
      if not SUBSEQUENCE(L1, G2.VERTICES) then
        return FALSE;
      end if;
      while NON_EMPTY(L2) loop
        if not IS_LINK(VALUE(VALUE(L2).THE_FIRST_OP_ID),
                       VALUE(VALUE(L2).THE_SECOND_OP_ID), G2) then
          return FALSE;
        end if;
        NEXT(L2);
      end loop;
      return TRUE;
    end SUB_SET;
  begin
    -- equal_graphs
    return (SUB_SET(G1, G2) and SUB_SET(G2, G1));
  end EQUAL_GRAPHS;

  procedure EMPTY(G : out GRAPH) is
  begin
    EMPTY(G.VERTICES);
    EMPTY(G.LINKS);
  end EMPTY;
```

```
function IS_NODE(X : in VERTEX; G : GRAPH) return BOOLEAN is
begin
  if LOOK4(X, G.VERTICES) /= null then
    return TRUE;
  else
    return FALSE;
  end if;
end IS_NODE;

function IS_LINK(X : in VERTEX; Y : in VERTEX; G : in GRAPH) return BOOLEAN is
  L : E_LISTS.LIST := G.LINKS;
begin
  while L /= null loop
    if VALUE(VALUE(L).THE_FIRST_OP_ID) = X and
          VALUE(VALUE(L).THE_SECOND_OP_ID) = Y then
      return TRUE;
    end if;
    L := L.NEXT;
  end loop;
  return FALSE;
end IS_LINK;

procedure ADD(X : in VERTEX; G : in out GRAPH) is
begin
  ADD(X, G.VERTICES);
end ADD;

procedure ADD(L : in LINK_DATA; G : in out GRAPH) is
begin
  if LOOK4(L.THE_FIRST_OP_ID.ELEMENT, G.VERTICES) /= null and
     LOOK4(L.THE_SECOND_OP_ID.ELEMENT, G.VERTICES) /= null then

    ADD(L, G.LINKS);

  end if;
end ADD;

procedure REMOVE(X : in VERTEX; G : in out GRAPH) is
  S : V_LISTS.LIST;
  L : V_LISTS.LIST;
  PREV : V_LISTS.LIST := null;
begin
  SCAN_CHILDREN(X, G, S);
  while NON_EMPTY(S) loop
    REMOVE(X, VALUE(S), G);
    NEXT(S);
  end loop;
  SCAN_PARENTS(X, G, S);
  while NON_EMPTY(S) loop
    REMOVE(VALUE(S), X, G);
```

```
      NEXT(S);
    end loop;
    REMOVE(X, G.VERTICES);
  end REMOVE;


procedure REMOVE(X : in VERTEX; Y : in VERTEX; G : in out GRAPH) is
  L : E_LISTS.LIST := G.LINKS;
  PREV : E_LISTS.LIST := null;
  TEMP : E_LISTS.LIST := null;
begin
  while NON_EMPTY(L) loop
    if VALUE(VALUE(L).THE_FIRST_OP_ID) = X and
              VALUE(VALUE(L).THE_SECOND_OP_ID) = Y then
      TEMP := L;
      NEXT(L);
      FREE(TEMP);
      if PREV /= null then
        PREV.NEXT := L;
      else
        G.LINKS := L;
      end if;
    else
      PREV := L;
      NEXT(L);
    end if;
  end loop;
end REMOVE;


procedure SCAN_NODES(G : in GRAPH; S : in out V_LISTS.LIST) is
  L : V_LISTS.LIST := G.VERTICES;
begin
  EMPTY(S);
  while NON_EMPTY(L) loop
    ADD(VALUE(L), S);
    NEXT(L);
  end loop;
end SCAN_NODES;


procedure SCAN_PARENTS(X : in VERTEX; G : in GRAPH;
                                      S : in out V_LISTS.LIST) is
  L : E_LISTS.LIST := G.LINKS;

begin
  EMPTY(S);
  while NON_EMPTY(L) loop
    if VALUE(VALUE(L).THE_SECOND_OP_ID) = X then
      ADD(VALUE(VALUE(L).THE_FIRST_OP_ID), S);
    end if;
    NEXT(L);
  end loop;
end SCAN_PARENTS;
```

```
   procedure SCAN_CHILDREN(X : in VERTEX; G : in GRAPH;
                                          S : in out V_LISTS.LIST) is
      L : E_LISTS.LIST := G.LINKS;
   begin
      EMPTY(S);
      while NON_EMPTY(L) loop
        if VALUE(VALUE(L).THE_FIRST_OP_ID) = X then
          ADD(VALUE(VALUE(L).THE_SECOND_OP_ID), S);
        end if;
        NEXT(L);
      end loop;
   end SCAN_CHILDREN;

   procedure DUPLICATE(G1 : in GRAPH; G2 : in out GRAPH) is
   begin
      DUPLICATE(G1.VERTICES, G2.VERTICES);
      DUPLICATE(G1.LINKS, G2.LINKS);
   end DUPLICATE;

   procedure T_SORT(G : in GRAPH; S : in out V_LISTS.LIST) is
      G1 : GRAPH;
      T, L, P : V_LISTS.LIST;
   begin
      EMPTY(T);
      DUPLICATE(G, G1);
      SCAN_NODES(G1, L);
      while NON_EMPTY(L) loop
        SCAN_PARENTS(VALUE(L), G1, P);
        if not NON_EMPTY(P) then
          ADD(VALUE(L), T);
          REMOVE(VALUE(L), G1);
          SCAN_NODES(G1, L);
        else
          NEXT(L);
        end if;
      end loop;
      SCAN_NODES(G1, L);
      if NON_EMPTY(L) then
        EMPTY(S);
      else
        LIST_REVERSE(T, S);
      end if;
   end T_SORT;
end GRAPHS;
```

APPENDIX X STATIC SCHEDULER HARMONIC BLOCK BUILDER

```
--------------------------------------------------------------------
-- file:       hbb_s.a
-- author:     murat kilic
-- date:       sep 89
-- modified:   oct 89 by murat kilic
--------------------------------------------------------------------


with FILES; use FILES;
package HARMONIC_BLOCK_BUILDER is

  procedure CALC_PERIODIC_EQUIVALENTS
          (OPT_LIST : in out DIGRAPH.V_LISTS.LIST);

  procedure FIND_BASE_BLOCK
          (PRECEDENCE_LIST : in DIGRAPH.V_LISTS.LIST;
           BASE_BLOCK       : out VALUE );

  procedure FIND_BLOCK_LENGTH
          (PRECEDENCE_LIST : in DIGRAPH.V_LISTS.LIST;
           HARMONIC_BLOCK_LENGTH  : out INTEGER );

  NO_BASE_BLOCK               : exception;
  NO_OPERATOR_IN_LIST         : exception;
  MET_NOT_LESS_THAN_PERIOD : exception;

end HARMONIC_BLOCK_BUILDER;
```

APPENDIX Y STATIC SCHEDULER HARMONIC BLOCK BUILDER

```
-------------------------------------------------------------
-- file:     hbb_b.a
-- author:   murat kilic
-- date:     sep 89
-- modified: oct 89
-------------------------------------------------------------


with TEXT_IO;
with FILES; use FILES;
package body HARMONIC_BLOCK_BUILDER is


   procedure CALC_PERIODIC_EQUIVALENTS
            (OPT_LIST : in out DIGRAPH.V_LISTS.LIST) is

     V : DIGRAPH.V_LISTS.LIST := OPT_LIST;

     procedure VERIFY_1 (O : in OPERATOR) is

     -- Check to ensure that MRT has a value for sporadic operators
     begin
       if O.THE_MRT >= O.THE_PERIOD then
         Exception_Operator := O.THE_OPERATOR_ID;
         raise MRT_NOT_LESS_THAN_PERIOD;
       end if;
     end VERIFY_1;

     procedure CALCULATE_NEW_PERIOD (O : in out OPERATOR) is
       DIFFERENCE : VALUE;
package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);
     begin
       DIFFERENCE := O.THE_MRT - O.THE_MET;
       if DIFFERENCE < O.THE_MCP then
         O.THE_PERIOD := DIFFERENCE;
       else
         O.THE_PERIOD := O.THE_MCP;
       end if;
     TEXT_IO.put("The new PERIOD is => ");
     VALUE_IO.put(O.THE_PERIOD);
     TEXT_IO.NEW_LINE;
     end CALCULATE_NEW_PERIOD;
```

```
begin -- main CALC_PERIODIC_EQUIVALENTS
  while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
    if DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD = 0 then
      CALCULATE_NEW_PERIOD(V.ELEMENT);
    end if;
    VERIFY_1(DIGRAPH.V_LISTS.VALUE(V));
    DIGRAPH.V_LISTS.NEXT(V);
  end loop;
end CALC_PERIODIC_EQUIVALENTS;

procedure FIND_BASE_BLOCK (PRECEDENCE_LIST : in DIGRAPH.V_LISTS.LIST;
                           BASE_BLOCK      : out VALUE ) is

  P_LIST : DIGRAPH.V_LISTS.LIST := PRECEDENCE_LIST;
  DIVISOR : VALUE;
  ALTERNATE_SEQUENCE  : DIGRAPH.V_LISTS.LIST;
  BASE_BLOCK_SEQUENCE : DIGRAPH.V_LISTS.LIST;
package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);

  function FIND_MINIMUM_PERIOD (P_LIST : in DIGRAPH.V_LISTS.LIST)
                                                  return VALUE is

    V : DIGRAPH.V_LISTS.LIST := P_LIST;
    MIN_PERIOD : VALUE := 0;

  begin
    if DIGRAPH.V_LISTS.NON_EMPTY(V) then
      MIN_PERIOD := DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD;
      DIGRAPH.V_LISTS.NEXT(V);
      while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
        if DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD < MIN_PERIOD then
          MIN_PERIOD := DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD;
        end if;
        DIGRAPH.V_LISTS.NEXT(V);
      end loop;
      return MIN_PERIOD;
    else
      raise NO_OPERATOR_IN_LIST;
    end if;
  end FIND_MINIMUM_PERIOD;

  function MODE_DIVIDE (THE_PERIOD : in VALUE) return VALUE is
  begin
    return (THE_PERIOD mod DIVISOR);
  end MODE_DIVIDE;

  procedure INITIAL_PASS (P_LIST : in out DIGRAPH.V_LISTS.LIST;
          BASE_BLOCK_SEQUENCE : in out DIGRAPH.V_LISTS.LIST;
          ALTERNATE_SEQUENCE  : in out DIGRAPH.V_LISTS.LIST) is

    ORIG_SEQUENCE : DIGRAPH.V_LISTS.LIST := P_LIST;
    OP_FROM_ORG_SEQ : OPERATOR;
```

```
        REMAINDER : VALUE;
        THE_PERIOD : VALUE;
      begin
        while DIGRAPH.V_LISTS.NON_EMPTY(ORIG_SEQUENCE) loop
          THE_PERIOD := DIGRAPH.V_LISTS.VALUE(ORIG_SEQUENCE).THE_PERIOD;
          REMAINDER := MODE_DIVIDE (THE_PERIOD);
          OP_FROM_ORG_SEQ := DIGRAPH.V_LISTS.VALUE(ORIG_SEQUENCE);
          if REMAINDER = 0 then
            DIGRAPH.V_LISTS.ADD (OP_FROM_ORG_SEQ, BASE_BLOCK_SEQUENCE);
          else
            DIGRAPH.V_LISTS.ADD (OP_FROM_ORG_SEQ, ALTERNATE_SEQUENCE);
          end if;
          DIGRAPH.V_LISTS.NEXT(ORIG_SEQUENCE);
        end loop;
      end INITIAL_PASS;

    begin -- main FIND_BASE_BLOCK
      DIVISOR := FIND_MINIMUM_PERIOD(P_LIST);
      INITIAL_PASS(P_LIST, BASE_BLOCK_SEQUENCE, ALTERNATE_SEQUENCE);
      while DIGRAPH.V_LISTS.NON_EMPTY(ALTERNATE_SEQUENCE) loop
        if DIVISOR = 1 then
          raise NO_BASE_BLOCK;
          -- exit and terminate the Static Scheduler
        else
          DIVISOR := DIVISOR - 1;
          ALTERNATE_SEQUENCE  := null;
          BASE_BLOCK_SEQUENCE := null;
          INITIAL_PASS(P_LIST, BASE_BLOCK_SEQUENCE, ALTERNATE_SEQUENCE);
        end if;
      end loop;
      BASE_BLOCK := DIVISOR;
    end FIND_BASE_BLOCK;


    procedure FIND_BLOCK_LENGTH
              (PRECEDENCE_LIST        : in DIGRAPH.V_LISTS.LIST;
               HARMONIC_BLOCK_LENGTH : out INTEGER ) is

      ORIG_SEQUENCE : DIGRAPH.V_LISTS.LIST := PRECEDENCE_LIST;
      NUMBER1    : VALUE;
      NUMBER2    : VALUE;
      LCM        : VALUE;
      GCD        : VALUE;
      TARGET_NO : VALUE;

      function FIND_GCD
              (NUMBER1 : in VALUE; NUMBER2 : in VALUE)
               return VALUE is NEW_GCD : VALUE;
      begin
        while GCD /= 0 loop
          if (NUMBER1 mod GCD = 0) and (NUMBER2 mod GCD = 0) then
            NEW_GCD := GCD;
```

313

```
              return NEW_GCD;
            else
              GCD := GCD - 1;
            end if;
          end loop;
        end FIND_GCD;

        function FIND_LCM (NUMBER1, NUMBER2 : VALUE) return VALUE is
        begin
          return(NUMBER1 * NUMBER2) / GCD;
         end FIND_LCM;

    begin -- main FIND_BLOCK_LENGTH
      if DIGRAPH.V_LISTS.NON_EMPTY(ORIG_SEQUENCE) then
        NUMBER1 := DIGRAPH.V_LISTS.VALUE(ORIG_SEQUENCE).THE_PERIOD;
        DIGRAPH.V_LISTS.NEXT(ORIG_SEQUENCE);
        while DIGRAPH.V_LISTS.NON_EMPTY(ORIG_SEQUENCE) loop
          NUMBER2 := DIGRAPH.V_LISTS.VALUE(ORIG_SEQUENCE).THE_PERIOD;
          if NUMBER2 > NUMBER1 then
            GCD := NUMBER1;
            TARGET_NO := NUMBER2;
          else
            GCD := NUMBER2;
            TARGET_NO := NUMBER1;
          end if;
          GCD := FIND_GCD(GCD, TARGET_NO);
          LCM := FIND_LCM(NUMBER1, NUMBER2);
          NUMBER1 := LCM;
          DIGRAPH.V_LISTS.NEXT(ORIG_SEQUENCE);
        end loop;
        HARMONIC_BLOCK_LENGTH := LCM;
      else
        raise NO_OPERATOR_IN_LIST;
      end if;
    end FIND_BLOCK_LENGTH;

end HARMONIC_BLOCK_BUILDER;
```

APPENDIX Z STATIC SCHEDULER ALGORITHMS

```
-------------------------------------------------------------------
-- file:      scheduler_s.a
-- author:    murat kilic
-- date:      dec 89
-- modified:  dec 89 by laura j. white
-------------------------------------------------------------------


with FILES; use FILES;
package OPERATOR_SCHEDULER is

  procedure TEST_DATA
            (INPUT_LIST            : in DIGRAPH.V_LISTS.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER);

  procedure SCHEDULE_INITIAL_SET
            (PRECEDENCE_LIST       : in DIGRAPH.V_LISTS.LIST;
             THE_SCHEDULE_INPUTS   : in out SCHEDULE_INPUTS_LIST.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER;
             STOP_TIME             : in out INTEGER);

  procedure SCHEDULE_REST_OF_BLOCK
            (PRECEDENCE_LIST       : in DIGRAPH.V_LISTS.LIST;
             THE_SCHEDULE_INPUTS   : in out SCHEDULE_INPUTS_LIST.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER;
             STOP_TIME             : in INTEGER);

  procedure SCHEDULE_WITH_EARLIEST_START
            (THE_GRAPH             : in DIGRAPH.GRAPH;
             AGENDA                : in out SCHEDULE_INPUTS_LIST.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER);

  procedure SCHEDULE_WITH_EARLIEST_DEADLINE
            (THE_GRAPH             : in DIGRAPH.GRAPH;
             AGENDA                : in out SCHEDULE_INPUTS_LIST.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER);

  procedure CREATE_STATIC_SCHEDULE
            (THE_GRAPH             : in DIGRAPH.GRAPH;
             THE_SCHEDULE_INPUTS   : in SCHEDULE_INPUTS_LIST.LIST;
             HARMONIC_BLOCK_LENGTH : in INTEGER);

  MISSED_DEADLINE  : exception;
  OVER_TIME        : exception;
```

```
MISSED_OPERATOR   : exception;

end OPERATOR_SCHEDULER;
```

APPENDIX AA STATIC SCHEDULER ALGORITHMS

```
---------------------------------------------------------------
- file:      scheduler_b.a
-- author:    murat kilic
-- date:      nov 89
-- modified:  dec 89 by laura j. white
---------------------------------------------------------------


with FILES; use FILES;
with TEXT_IO;
package body OPERATOR_SCHEDULER is

  procedure TEST_DATA  (INPUT_LIST           : in DIGRAPH.V_LISTS.LIST;
                        HARMONIC_BLOCK_LENGTH : in INTEGER) is

    procedure CALC_TOTAL_TIME (INPUT_LIST           : in DIGRAPH.V_LISTS.LIST;
                               HARMONIC_BLOCK_LENGTH : in INTEGER) is
      V : DIGRAPH.V_LISTS.LIST := INPUT_LIST;
      TIMES      : FLOAT := 0.0;
      OP_TIME    : FLOAT := 0.0;
      TOTAL_TIME : FLOAT := 0.0;
      PER        : OPERATOR;
      BAD_TOTAL_TIME : exception;

      function CALC_NO_OF_PERIODS (HARMONIC_BLOCK_LENGTH : in INTEGER;
                                   THE_PERIOD : in INTEGER) return FLOAT is
      begin
        return FLOAT(HARMONIC_BLOCK_LENGTH) / FLOAT(THE_PERIOD);
      end CALC_NO_OF_PERIODS;

      function MULTIPLY_BY_MET (TIMES   : in FLOAT;
                                THE_MET : in VALUE) return FLOAT is
      begin
        return TIMES * FLOAT(THE_MET);
      end MULTIPLY_BY_MET;

      function ADD_TO_SUM (OP_TIME : in FLOAT) return FLOAT is
      begin
        return TOTAL_TIME + OP_TIME;
      end ADD_TO_SUM;

    begin --main CALC_TOTAL_TIME
      while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
        PER := DIGRAPH.V_LISTS.VALUE(V);
```

317

```
      TIMES:= CALC_NO_OF_PERIODS (HARMONIC_BLOCK_LENGTH , PER.THE_PERIOD);
      OP_TIME := MULTIPLY_BY_MET (TIMES, DIGRAPH.V_LISTS.VALUE(V).THE_MET);
      TOTAL_TIME := ADD_TO_SUM (OP_TIME);
      if TOTAL_TIME > FLOAT(HARMONIC_BLOCK_LENGTH) then
        raise BAD_TOTAL_TIME;
      else
        DIGRAPH.V_LISTS.NEXT(V);
      end if;
    end loop;
    exception
      when BAD_TOTAL_TIME =>
          TEST_VERIFIED := FALSE;
          TEXT_IO.PUT("The total execution time of the operators exceeds");
          TEXT_IO.PUT_LINE(" the HARMONIC_BLOCK_LENGTH");
          TEXT_IO.NEW_LINE;
end CALC_TOTAL_TIME;

procedure CALC_HALF_PERIODS (INPUT_LIST : in DIGRAPH.V_LISTS.LIST) is

  V : DIGRAPH.V_LISTS.LIST := INPUT_LIST;
  HALF_PERIOD : FLOAT;
  FAIL_HALF_PERIOD : exception;

  function DIVIDE_PERIOD_BY_2 (THE_PERIOD : in VALUE) return FLOAT is
  begin
    return FLOAT(THE_PERIOD) / 2.0;
  end DIVIDE_PERIOD_BY_2;

begin --main CALC_HALF_PERIODS;
  while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
    HALF_PERIOD := DIVIDE_PERIOD_BY_2(DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD);
    if FLOAT(DIGRAPH.V_LISTS.VALUE(V).THE_MET) > HALF_PERIOD then
      Exception_Operator := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
      raise FAIL_HALF_PERIOD;
    else
      DIGRAPH.V_LISTS.NEXT(V);
    end if;
  end loop;
  exception
    when FAIL_HALF_PERIOD =>
        TEST_VERIFIED := FALSE;
        TEXT_IO.PUT ("The MET of Operator ");
        VARSTRING.PUT (Exception_Operator);
        TEXT_IO.PUT_LINE (" is greater than half of its period.");
end CALC_HALF_PERIODS;

procedure CALC_RATIO_SUM (INPUT_LIST : in DIGRAPH.V_LISTS.LIST) is
  V : DIGRAPH.V_LISTS.LIST := INPUT_LIST;
  RATIO : FLOAT;
  RATIO_SUM : FLOAT := 0.0;
  THE_MET : VALUE;
```

```
      THE_PERIOD : VALUE;
      RATIO_TOO_BIG : exception;

      function DIVIDE_MET_BY_PERIOD (THE_MET : in VALUE;
                                     THE_PERIOD : in VALUE) return FLOAT is
      begin
        return FLOAT(THE_MET) / FLOAT(THE_PERIOD);
      end DIVIDE_MET_BY_PERIOD;

      function ADD_TO_TIME (RATIO : in FLOAT) return FLOAT is
      begin
        return RATIO_SUM + RATIO;
      end ADD_TO_TIME;

   begin --main CALC_RATIO_SUM
     while DIGRAPH.V_LISTS.NON_EMPTY(V)  loop
       THE_MET :=  DIGRAPH.V_LISTS.VALUE(V).THE_MET;
       THE_PERIOD := DIGRAPH.V_LISTS.VALUE(V).THE_PERIOD;
       RATIO := DIVIDE_MET_BY_PERIOD(THE_MET,THE_PERIOD);
       RATIO_SUM := ADD_TO_TIME(RATIO);
       DIGRAPH.V_LISTS.NEXT(V);
     end loop;
     if RATIO_SUM > 0.5 then
       raise RATIO_TOO_BIG;
     end if;
     exception
       when RATIO_TOO_BIG =>
            TEST_VERIFIED := FALSE;
            TEXT_IO.PUT ("The total MET/PERIOD ratio sum of operators is");
            TEXT_IO.PUT_LINE (" greater than 0.5");
   end CALC_RATIO_SUM;

  begin --main TEST_DATA
    CALC_TOTAL_TIME(INPUT_LIST, HARMONIC_BLOCK_LENGTH);
    CALC_HALF_PERIODS(INPUT_LIST);
    CALC_RATIO_SUM(INPUT_LIST);
  end TEST_DATA;
--------------------------------------------------------
  procedure VERIFY_TIME_LEFT (HARMONIC_BLOCK_LENGTH : in INTEGER;
                              STOP_TIME : in INTEGER) is
  begin
    if STOP_TIME > HARMONIC_BLOCK_LENGTH then
      raise OVER_TIME;
      --exit and terminate the Static Scheduler
    end if;
  end VERIFY_TIME_LEFT;
--------------------------------------------------------
  procedure CREATE_INTERVAL (THE_OPERATOR : in OPERATOR;
                             INPUT        : in out SCHEDULE_INPUTS;
                             OLD_LOWER    : in VALUE) is
    LOWER_BOUND : VALUE;
```

319

```
function CALC_LOWER_BOUND return VALUE is
begin
  -- since CREATE_INTERVAL function is used in both SCHEDULE_INITIAL_SET
  -- and SCHEDULE_REST_OF_BLOCK (OLD_LOWER /= 0) check is needed. In
  -- case of the operator is scheduled somewhere in its interval and
  -- (OLD_LOWER /= 0),
  -- this check guarantees that the periods will be consistent.
  if (OLD_LOWER /= 0) and (OLD_LOWER < INPUT.THE_START) then
    LOWER_BOUND := OLD_LOWER + THE_OPERATOR.THE_PERIOD;
  else
    LOWER_BOUND := INPUT.THE_START + THE_OPERATOR.THE_PERIOD;
  end if;
  return LOWER_BOUND;
end CALC_LOWER_BOUND;

function CALC_UPPER_BOUND return VALUE is
begin
  if THE_OPERATOR.THE_WITHIN = 0 then
    return LOWER_BOUND + THE_OPERATOR.THE_PERIOD - THE_OPERATOR.THE_MET;
  -- if the operator has a WITHIN constraint, the upper bound of the
  -- interval is reduced.
  else
    return LOWER_BOUND + THE_OPERATOR.THE_WITHIN - THE_OPERATOR.THE_MET;
  end if;
end CALC_UPPER_BOUND;
begin --main CREATE_INTERVAL
  INPUT.THE_LOWER := CALC_LOWER_BOUND;
  INPUT.THE_UPPER := CALC_UPPER_BOUND;
end CREATE_INTERVAL;
----------------------------------------------------
procedure SCHEDULE_INITIAL_SET
                  (PRECEDENCE_LIST        : in DIGRAPH.V_LISTS.LIST;
                   THE_SCHEDULE_INPUTS    : in out SCHEDULE_INPUTS_LIST.LIST;
                   HARMONIC_BLOCK_LENGTH  : in INTEGER;
                   STOP_TIME              : in out INTEGER) is

  V : DIGRAPH.V_LISTS.LIST := PRECEDENCE_LIST;
  START_TIME : INTEGER := 0;
  NEW_INPUT  : SCHEDULE_INPUTS;
  OLD_LOWER  : VALUE := 0;

begin --SCHEDULE_INITIAL_SET
  while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
    Exception_Operator := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
    NEW_INPUT.THE_OPERATOR := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
    NEW_INPUT.THE_START := START_TIME;
    STOP_TIME := START_TIME + DIGRAPH.V_LISTS.VALUE(V).THE_MET;
    VERIFY_TIME_LEFT(HARMONIC_BLOCK_LENGTH, STOP_TIME);
    NEW_INPUT.THE_STOP := STOP_TIME;
    START_TIME := STOP_TIME;
```

320

```
          -- for every operator in SCHEDULE_INITIAL_SET, OLD_LOWER is zero.
          -- So we always send zero value to CREATE_INTERVAL.
          CREATE_INTERVAL(DIGRAPH.V_LISTS.VALUE(V), NEW_INPUT, OLD_LOWER);
          SCHEDULE_INPUTS_LIST.ADD (NEW_INPUT, THE_SCHEDULE_INPUTS);
          DIGRAPH.V_LISTS.NEXT(V);
      end loop;
    end SCHEDULE_INITIAL_SET;
  -----------------------------------------------------
    procedure SCHEDULE_REST_OF_BLOCK
                  (PRECEDENCE_LIST:in DIGRAPH.V_LISTS.LIST;
                   THE_SCHEDULE_INPUTS     : in out SCHEDULE_INPUTS_LIST.LIST;
                   HARMONIC_BLOCK_LENGTH : in INTEGER;
                   STOP_TIME             : in INTEGER) is

      V : DIGRAPH.V_LISTS.LIST := PRECEDENCE_LIST;
      TEMP : SCHEDULE_INPUTS_LIST.LIST := THE_SCHEDULE_INPUTS;
      V_LIST  : DIGRAPH.V_LISTS.LIST;
      P : SCHEDULE_INPUTS_LIST.LIST;
      S : SCHEDULE_INPUTS_LIST.LIST;
      START_TIME : INTEGER := 0;
      TIME_STOP  : INTEGER := STOP_TIME;
      NEW_INPUT  : SCHEDULE_INPUTS;
      OLD_LOWER  : VALUE;


    begin
      DIGRAPH.V_LISTS.DUPLICATE(PRECEDENCE_LIST, V_LIST);

      SCHEDULE_INPUTS_LIST.LIST_REVERSE(THE_SCHEDULE_INPUTS, P);

    loop
      while SCHEDULE_INPUTS_LIST.NON_EMPTY(P) loop
        if SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER < HARMONIC_BLOCK_LENGTH then
            NEW_INPUT.THE_OPERATOR := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
          -- check if the operator can be scheduled in its interval
          if SCHEDULE_INPUTS_LIST.VALUE(P).THE_UPPER - TIME_STOP
                                    >= DIGRAPH.V_LISTS.VALUE(V).THE_MET then
            if SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER >= TIME_STOP then
              START_TIME := SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER;
            else
              START_TIME := TIME_STOP;
            end if;
              NEW_INPUT.THE_START := START_TIME;
              NEW_INPUT.THE_STOP := START_TIME + DIGRAPH.V_LISTS.VALUE(V).THE_MET;
            TIME_STOP := NEW_INPUT.THE_STOP;
            OLD_LOWER := SCHEDULE_INPUTS_LIST.VALUE(P).THE_LOWER;
              CREATE_INTERVAL(DIGRAPH.V_LISTS.VALUE(V), NEW_INPUT, OLD_LOWER);
              SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT, TEMP);
              SCHEDULE_INPUTS_LIST.ADD(NEW_INPUT, S);
            Exception_Operator := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
              VERIFY_TIME_LEFT(HARMONIC_BLOCK_LENGTH, TIME_STOP);
```

321

```
            DIGRAPH.V_LISTS.NEXT(V);
                SCHEDULE_INPUTS_LIST.NEXT(P);
         -- if the operator can not be scheduled in its interval raise the
         -- exception
            else
            Exception_Operator := DIGRAPH.V_LISTS.VALUE(V).THE_OPERATOR_ID;
              raise MISSED_DEADLINE;
            end if;
      else
         DIGRAPH.V_LISTS.REMOVE(DIGRAPH.V_LISTS.VALUE(V), V_LIST);
         DIGRAPH.V_LISTS.NEXT(V);
            SCHEDULE_INPUTS_LIST.NEXT(P);
      end if;
    end loop;
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(S) then
       SCHEDULE_INPUTS_LIST.LIST_REVERSE(S, P);
       SCHEDULE_INPUTS_LIST.EMPTY(S);
       V := V_LIST;
    else
       exit;
    end if;
  end loop;
  SCHEDULE_INPUTS_LIST.LIST_REVERSE(TEMP, THE_SCHEDULE_INPUTS);

end SCHEDULE_REST_OF_BLOCK;
-----------------------------------------------------
procedure BUILD_OP_INFO_LIST
             (THE_GRAPH        : in DIGRAPH.GRAPH;
              THE_OP_INFO_LIST : in out OP_INFO_LIST.LIST) is
    -- this procedure finds each operator's successors and predecessors
    -- first and creates the OPERATOR_INFO_LIST.
    V : DIGRAPH.V_LISTS.LIST := THE_GRAPH.VERTICES;
    S : DIGRAPH.V_LISTS.LIST;
    P : DIGRAPH.V_LISTS.LIST;
    NEW_NODE : OP_INFO;

  begin
    while DIGRAPH.V_LISTS.NON_EMPTY(V) loop
      DIGRAPH.SCAN_CHILDREN(DIGRAPH.V_LISTS.VALUE(V), THE_GRAPH, S);
      DIGRAPH.SCAN_PARENTS(DIGRAPH.V_LISTS.VALUE(V), THE_GRAPH, P);
      NEW_NODE.NODE := DIGRAPH.V_LISTS.VALUE(V);
      NEW_NODE.SUCCESSORS := S;
      NEW_NODE.PREDICESSORS := P;
      OP_INFO_LIST.ADD(NEW_NODE, THE_OP_INFO_LIST);
      DIGRAPH.V_LISTS.NEXT(V);
    end loop;
  end BUILD_OP_INFO_LIST;
-----------------------------------------------------
  procedure PROCESS_EST_END_NODE
                 (MAY_BE_AVAILABLE: in out SCHEDULE_INPUTS_LIST.LIST;
                  OPT             : in OPERATOR) is
```

```
              -- transfer the OPERATOR record into SCHEDULE_INFO record and adds
              -- that into the MAY_AVAILABLE_LIST for the Earliest Start Scheduling
              -- Algorithm.  Initially all the values are zero.
              NEW_NODE : SCHEDULE_INPUTS;

          begin
              NEW_NODE.THE_OPERATOR := OPT.THE_OPERATOR_ID;
              SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, MAY_BE_AVAILABLE);
          end PROCESS_EST_END_NODE;
--------------------------------------------------------------
          procedure PROCESS_EDL_END_NODE
                              (MAY_BE_AVAILABLE: in out SCHEDULE_INPUTS_LIST.LIST;
                               OPT               : in OPERATOR) is
              --transfer the OPERATOR record into SCHEDULE_INFO record and adds that
              --into the MAY_AVAILABLE_LIST for the Earliest Deadline Scheduling
              -- Algorithm.  Initially all the values are zero.
              NEW_NODE : SCHEDULE_INPUTS;

          begin
              NEW_NODE.THE_OPERATOR := OPT.THE_OPERATOR_ID;
              NEW_NODE.THE_LOWER := 0; -- we can omit this, because it's already zero.
              if OPT.THE_WITHIN /= 0 then
                NEW_NODE.THE_UPPER := OPT.THE_WITHIN;
              else
                NEW_NODE.THE_UPPER := OPT.THE_PERIOD;
              end if;
              SCHEDULE_INPUTS_LIST.ADD(NEW_NODE, MAY_BE_AVAILABLE);
          end PROCESS_EDL_END_NODE;
--------------------------------------------------------------
          function FIND_OPERATOR(THE_OP_INFO_LIST : in OP_INFO_LIST.LIST;
                                 ID                : in OPERATOR_ID)
                                 return OP_INFO_LIST.LIST is
              -- finds the operator that we use currently to get the required information.
              TEMP : OP_INFO_LIST.LIST := THE_OP_INFO_LIST;

          -- assumed that it's guaranteed to find an operator.
          begin
              while OP_INFO_LIST.NON_EMPTY(TEMP) loop
                if VARSTRING.EQUAL(OP_INFO_LIST.VALUE(TEMP).NODE.THE_OPERATOR_ID, ID) then
                  return TEMP ;
                end if;
                OP_INFO_LIST.NEXT(TEMP);
              end loop;
          end FIND_OPERATOR;
-------------------------------,----------------------------------
          function CHECK_AGENDA(THE_NODE : in OP_INFO;
                                AGENDA    : in SCHEDULE_INPUTS_LIST.LIST)
                                return BOOLEAN is
              -- checks the AGENDA list to see if all the predecessors of the
              -- operator are in there.
              P   : DIGRAPH.V_LISTS.LIST := THE_NODE.PREDICESSORS;
```

```
      A   : SCHEDULE_INPUTS_LIST.LIST := AGENDA;
      OK : BOOLEAN := FALSE;
   begin
     while DIGRAPH.V_LISTS.NON_EMPTY(P) loop
        while SCHEDULE_INPUTS_LIST.NON_EMPTY(A) loop
          if VARSTRING.EQUAL(DIGRAPH.V_LISTS.VALUE(P).THE_OPERATOR_ID,
                            SCHEDULE_INPUTS_LIST.VALUE(A).THE_OPERATOR) then
             OK := TRUE;
             exit;
          end if;
          SCHEDULE_INPUTS_LIST.NEXT(A);
        end loop;
        if OK then
          DIGRAPH.V_LISTS.NEXT(P);
          A := AGENDA;
          OK := FALSE;
        else
          -- if the pointer reached to the end of the AGENDA, it means the
          -- operator is not in AGENDA, if so return FALSE.
          return OK;
        end if;
     end loop;
     -- if the pointer reached to the end of the predecessor list, it
     -- means the operator is in AGENDA.
     OK := TRUE;
     return OK;
   end CHECK_AGENDA;
------------------------------------------------------------
   procedure EST_INSERT
               (TARGET              : in SCHEDULE_INPUTS;
                MAY_BE_AVAILABLE : in out SCHEDULE_INPUTS_LIST.LIST) is
     -- used to insert the operators into the MAY_BE_AVAILABLE list to
     -- schedule for the Earliest Start Scheduling Algorithm.
     PREV : SCHEDULE_INPUTS_LIST.LIST := null;
     T    : SCHEDULE_INPUTS_LIST.LIST := MAY_BE_AVAILABLE;

   begin
     if NOT(SCHEDULE_INPUTS_LIST.NON_EMPTY(T)) then
        -- when MAY_BE_AVAILABLE list is empty, add the operator immediately.
        SCHEDULE_INPUTS_LIST.ADD(TARGET, MAY_BE_AVAILABLE);
     else

        -- in case the target operator's EST is smaller than the first operator's
        -- EST add the operator to the list immediately.
        if TARGET.THE_LOWER < SCHEDULE_INPUTS_LIST.VALUE(T).THE_LOWER then
           SCHEDULE_INPUTS_LIST.ADD(TARGET, MAY_BE_AVAILABLE);

        -- in case the operator with the same EST is in the list, do not insert,
        -- otherwise;  insert the operator in its order.
        elsif NOT(SCHEDULE_INPUTS_LIST.MEMBER(TARGET, MAY_BE_AVAILABLE)) then
           while SCHEDULE_INPUTS_LIST.NON_EMPTY(T) loop
```

324

```
          if TARGET.THE_LOWER > SCHEDULE_INPUTS_LIST.VALUE(T).THE_LOWER then
            PREV := T;
            SCHEDULE_INPUTS_LIST.NEXT(T);
          else
            exit;
          end if;
        end loop;
        SCHEDULE_INPUTS_LIST.ADD(TARGET, T);
        if SCHEDULE_INPUTS_LIST.NON_EMPTY(PREV) then
          PREV.NEXT := T;
        else
          MAY_BE_AVAILABLE := T;
        end if;
      end if;
    end if;
  end EST_INSERT;
---------------------------------------------------------
  procedure EDL_INSERT
              (TARGET               : in SCHEDULE_INPUTS;
               MAY_BE_AVAILABLE : in out SCHEDULE_INPUTS_LIST.LIST) is

    -- used to insert the operators into the MAY_BE_AVAILABLE list to
    -- schedule for the Earliest Deadline Scheduling Algorithm.
    PREV : SCHEDULE_INPUTS_LIST.LIST := null;
    T    : SCHEDULE_INPUTS_LIST.LIST := MAY_BE_AVAILABLE;

  begin
    if NOT(SCHEDULE_INPUTS_LIST.NON_EMPTY(T)) then
      SCHEDULE_INPUTS_LIST.ADD(TARGET, MAY_BE_AVAILABLE);
    else
      if TARGET.THE_UPPER < SCHEDULE_INPUTS_LIST.VALUE(T).THE_UPPER then
        SCHEDULE_INPUTS_LIST.ADD(TARGET, MAY_BE_AVAILABLE);
      elsif NOT(SCHEDULE_INPUTS_LIST.MEMBER(TARGET, MAY_BE_AVAILABLE)) then
        while SCHEDULE_INPUTS_LIST.NON_EMPTY(T) loop
          if TARGET.THE_UPPER > SCHEDULE_INPUTS_LIST.VALUE(T).THE_UPPER then
            PREV := T;
            SCHEDULE_INPUTS_LIST.NEXT(T);
          else
            exit;
          end if;
        end loop;
        SCHEDULE_INPUTS_LIST.ADD(TARGET, T);
        if SCHEDULE_INPUTS_LIST.NON_EMPTY(PREV) then
          PREV.NEXT := T;
        else
          MAY_BE_AVAILABLE := T;
        end if;
      end if;
    end if;
  end EDL_INSERT;
---------------------------------------------------------
```

```
function OPERATOR_IN_LIST(OPT_ID  : in OPERATOR_ID;
                          IN_LIST  : in SCHEDULE_INPUTS_LIST.LIST)
                          return BOOLEAN is
-- this is used to check if the operators in successors list are
-- already in the complete MAY_BE_AVAILABLE list for both EST and
-- EDL algorithms.
TEMP : OPERATOR_ID;
L    : SCHEDULE_INPUTS_LIST.LIST := IN_LIST;

begin
  while SCHEDULE_INPUTS_LIST.NON_EMPTY(L) loop
    TEMP := SCHEDULE_INPUTS_LIST.VALUE(L).THE_OPERATOR;
    if  VARSTRING.EQUAL(TEMP, OPT_ID) then
      return TRUE;
    else
      SCHEDULE_INPUTS_LIST.NEXT(L);
    end if;
  end loop;
  return FALSE;
end OPERATOR_IN_LIST;
----------------------------------------------------------------
procedure EST_INSERT_SUCCESSORS_OF_OPT
              (THE_NODE          : in OP_INFO;
               STOP_TIME         : in VALUE;
               MAY_BE_AVAILABLE : in out SCHEDULE_INPUTS_LIST.LIST) is
  -- inserts the successors of the selected operator into
    -- MAY_BE_AVAILABLE list in their orders if they do not
    -- exist in the list.
  S      : DIGRAPH.V_LISTS.LIST := THE_NODE.SUCCESSORS;
  T      : OPERATOR;
  OPT    : OPERATOR := THE_NODE.NODE;
  TARGET : SCHEDULE_INPUTS;
begin
  while DIGRAPH.V_LISTS.NON_EMPTY(S) loop
    T := DIGRAPH.V_LISTS.VALUE(S);
    if NOT(OPERATOR_IN_LIST(T.THE_OPERATOR_ID, MAY_BE_AVAILABLE)) then
      TARGET.THE_OPERATOR := DIGRAPH.V_LISTS.VALUE(S).THE_OPERATOR_ID;
      TARGET.THE_LOWER := STOP_TIME;
      EST_INSERT(TARGET, MAY_BE_AVAILABLE);
    end if;
    DIGRAPH.V_LISTS.NEXT(S);
  end loop;
end EST_INSERT_SUCCESSORS_OF_OPT;
----------------------------------------------------------------
procedure EDL_INSERT_SUCCESSORS_OF_OPT
              (THE_NODE          : in OP_INFO;
               STOP_TIME         : in VALUE;
               COMPLETE_LIST     : in out SCHEDULE_INPUTS_LIST.LIST;
               MAY_BE_AVAILABLE : in out SCHEDULE_INPUTS_LIST.LIST) is
  -- inserts the successors of the selected operator into
    -- MAY_BE_AVAILABLE list in their orders if they do not exist in
```

```
                -- the list.
      S       : DIGRAPH.V_LISTS.LIST := THE_NODE.SUCCESSORS;
      T       : OPERATOR;
      OPT     : OPERATOR := THE_NODE.NODE;
      TARGET : SCHEDULE_INPUTS;
    begin
      while DIGRAPH.V_LISTS.NON_EMPTY(S) loop
        T := DIGRAPH.V_LISTS.VALUE(S);
        if NOT(OPERATOR_IN_LIST(T.THE_OPERATOR_ID, COMPLETE_LIST)) then
          TARGET.THE_OPERATOR := T.THE_OPERATOR_ID;
          TARGET.THE_LOWER := STOP_TIME;
          -- while we are adding the successors, the deadline of these operators
          -- are calculated by adding either their finish_within if exists, or
          -- period to the stop_time of the last operator.
          if T.THE_WITHIN /= 0 then
            TARGET.THE_UPPER := STOP_TIME + T.THE_WITHIN;
          else
            TARGET.THE_UPPER := STOP_TIME + T.THE_PERIOD;
          end if;
          EDL_INSERT(TARGET, MAY_BE_AVAILABLE);
        end if;
        DIGRAPH.V_LISTS.NEXT(S);
      end loop;
    end EDL_INSERT_SUCCESSORS_OF_OPT;
----------------------------------------------------------
    procedure PROCESS_EST_AGENDA
                    (THE_OP_INFO_LIST: in OP_INFO_LIST.LIST;
                     MAY_BE_AVAILABLE: in out SCHEDULE_INPUTS_LIST.LIST;
                     AGENDA          : in out SCHEDULE_INPUTS_LIST.LIST;
                     HARMONIC_BLOCK_LENGTH : in INTEGER) is

      -- process the MAY_BE_AVILABLE list to produce AGENDA list which is
      -- used to create a schedule for Earliest Start Scheduling Algorithm.
      V         : SCHEDULE_INPUTS_LIST.LIST := MAY_BE_AVAILABLE;
      D         : SCHEDULE_INPUTS_LIST.LIST;
      A         : SCHEDULE_INPUTS_LIST.LIST;
      TEMP      : OP_INFO_LIST.LIST;
      TARGET    : SCHEDULE_INPUTS;
      NEW_INPUT : SCHEDULE_INPUTS;
      THE_NODE  : OP_INFO;
      CONTINUE  : BOOLEAN;
      STOP_TIME : VALUE := 0;
      OPT       : SCHEDULE_INPUTS;
      EST       : INTEGER;

    begin
      while SCHEDULE_INPUTS_LIST.VALUE(V).THE_LOWER < HARMONIC_BLOCK_LENGTH loop
        -- no need to check if all the predicessors are in the AGENDA, because
        -- this is the first node and has no predecessors.
        OPT := SCHEDULE_INPUTS_LIST.VALUE(V);
        TEMP := FIND_OPERATOR(THE_OP_INFO_LIST, OPT.THE_OPERATOR);
```

327

```
          THE_NODE := OP_INFO_LIST.VALUE(TEMP);
          if OPT.THE_LOWER > 0 then
            CONTINUE := CHECK_AGENDA(THE_NODE, AGENDA);
          else
            CONTINUE := TRUE;
          end if;

          -- if the opt.is not an end node check if all its successors in AGENDA.
          -- if not, select the other operator and repeat the same procedure.
          while NOT CONTINUE loop
            SCHEDULE_INPUTS_LIST.NEXT(V);
            OPT := SCHEDULE_INPUTS_LIST.VALUE(V);
            TEMP := FIND_OPERATOR(THE_OP_INFO_LIST, OPT.THE_OPERATOR);
            THE_NODE := OP_INFO_LIST.VALUE(TEMP);
            if OPT.THE_LOWER > 0 then
              CONTINUE := CHECK_AGENDA(THE_NODE, AGENDA);
            else
              CONTINUE := TRUE;
            end if;
          end loop;
          TARGET := SCHEDULE_INPUTS_LIST.VALUE(V);
          SCHEDULE_INPUTS_LIST.REMOVE(TARGET, MAY_BE_AVAILABLE);
          Exception_Operator := TARGET.THE_OPERATOR;
          VERIFY_TIME_LEFT(HARMONIC_BLOCK_LENGTH, STOP_TIME);
          if TARGET.THE_LOWER > STOP_TIME then
            -- zero initially for the first one
            TARGET.THE_START := TARGET.THE_LOWER;
          else
            TARGET.THE_START := STOP_TIME;
          end if;
          STOP_TIME := TARGET.THE_START + THE_NODE.NODE.THE_MET;
          TARGET.THE_STOP := STOP_TIME;
          SCHEDULE_INPUTS_LIST.ADD(TARGET, AGENDA);
          EST := TARGET.THE_START + THE_NODE.NODE.THE_PERIOD;

          -- if the operator can be scheduled again, put it back into the
          -- MAY_BE_AVAILABLE list in its order with the new EST.
          NEW_INPUT.THE_OPERATOR := TARGET.THE_OPERATOR;
          NEW_INPUT.THE_LOWER := EST;
          EST_INSERT(NEW_INPUT, MAY_BE_AVAILABLE);
          EST_INSERT_SUCCESSORS_OF_OPT(THE_NODE, STOP_TIME, MAY_BE_AVAILABLE);
          V := MAY_BE_AVAILABLE;
        end loop;
      A := AGENDA;
      SCHEDULE_INPUTS_LIST.LIST_REVERSE(A, AGENDA);
    end PROCESS_EST_AGENDA;
------------------------------------------------------------------------
    procedure PROCESS_EDL_AGENDA
                    (THE_OP_INFO_LIST: in OP_INFO_LIST.LIST;
                     COMPLETE_LIST    : in out SCHEDULE_INPUTS_LIST.LIST;
                     AGENDA           : in out SCHEDULE_INPUTS_LIST.LIST;
```

328

```
                    HARMONIC_BLOCK_LENGTH : in INTEGER) is

    -- process the MAY_BE_AVILABLE list to produce AGENDA list which is
    -- used to create a schedule for Earliest Deadline Scheduling Algorithm.
    V          : SCHEDULE_INPUTS_LIST.LIST :- COMPLETE_LIST;
    TEMP       : SCHEDULE_INPUTS_LIST.LIST :- COMPLETE_LIST;
    A          : SCHEDULE_INPUTS_LIST.LIST;
    T          : OP_INFO_LIST.LIST;
    PREV       : SCHEDULE_INPUTS_LIST.LIST :- null;
    TARGET     : SCHEDULE_INPUTS;
    NEW_INPUT  : SCHEDULE_INPUTS;
    THE_NODE   : OP_INFO;
    CONTINUE   : BOOLEAN;
    STOP_TIME  : VALUE :- 0;
    OPT        : SCHEDULE_INPUTS;
    EST        : INTEGER;

begin
  while SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP) loop
    if SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_LOWER < HARMONIC_BLOCK_LENGTH then
      -- no need to check if all the predecessors are in the AGENDA
      OPT :- SCHEDULE_INPUTS_LIST.VALUE(V);
      T   :- FIND_OPERATOR(THE_OP_INFO_LIST, OPT.THE_OPERATOR);
      THE_NODE :- OP_INFO_LIST.VALUE(T);
      if OPT.THE_LOWER > 0 then

        -- when the earliest start time of the operator is not zero, we
        -- need to check if all the predecessors of the operator are in
        -- AGENDA. No check otherwise.
        CONTINUE :- CHECK_AGENDA(THE_NODE, AGENDA);
      else
        CONTINUE :- TRUE;
      end if;

      -- if the opt. is not an end node check if all its successors
          -- in AGENDA. if not, select the other operator and repeat
          -- the same procedure.
      while NOT CONTINUE loop
        SCHEDULE_INPUTS_LIST.NEXT(V);
        OPT :- SCHEDULE_INPUTS_LIST.VALUE(V);
        T   :- FIND_OPERATOR(THE_OP_INFO_LIST, OPT.THE_OPERATOR);
        THE_NODE :- OP_INFO_LIST.VALUE(T);
        if OPT.THE_LOWER > 0 then
          CONTINUE :- CHECK_AGENDA(THE_NODE, AGENDA);
        else
          CONTINUE :- TRUE;
        end if;
      end loop;
      TARGET :- SCHEDULE_INPUTS_LIST.VALUE(V);
      SCHEDULE_INPUTS_LIST.REMOVE(TARGET, TEMP);
      if SCHEDULE_INPUTS_LIST.NON_EMPTY(PREV) then
```

```
        PREV.NEXT := TEMP;
      else
        COMPLETE_LIST := TEMP;
      end if;
    Exception_Operator := TARGET.THE_OPERATOR;
    VERIFY_TIME_LEFT(HARMONIC_BLOCK_LENGTH, STOP_TIME);
    if TARGET.THE_LOWER > STOP_TIME then
          --zero initially for the first one
      TARGET.THE_START := TARGET.THE_LOWER;
    else
      TARGET.THE_START := STOP_TIME;
    end if;
    STOP_TIME := TARGET.THE_START + THE_NODE.NODE.THE_MET;
    TARGET.THE_STOP := STOP_TIME;
    SCHEDULE_INPUTS_LIST.ADD(TARGET, AGENDA);
    EST := TARGET.THE_START + THE_NODE.NODE.THE_PERIOD;
    NEW_INPUT.THE_OPERATOR := TARGET.THE_OPERATOR;
    NEW_INPUT.THE_LOWER := EST;

    if THE_NODE.NODE.THE_WITHIN /= 0 then
      NEW_INPUT.THE_UPPER := EST + THE_NODE.NODE.THE_WITHIN;
    else
      NEW_INPUT.THE_UPPER := EST + THE_NODE.NODE.THE_PERIOD;
    end if;
    EDL_INSERT(NEW_INPUT, TEMP);

    -- this is to keep track of the COMPLETE_LIST pointer
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(PREV) then
      -- the pointer is pointing a record other than first one.
      PREV.NEXT := TEMP;
    else
      -- the pointer is pointing the first record in the list.
      COMPLETE_LIST := TEMP;
    end if;

    EDL_INSERT_SUCCESSORS_OF_OPT
                      (THE_NODE, STOP_TIME, COMPLETE_LIST, TEMP);
    V := TEMP;

    -- this is to keep track of the COMPLETE_LIST pointer
    if SCHEDULE_INPUTS_LIST.NON_EMPTY(PREV) then
      -- the pointer is pointing a record other than first one.
      PREV.NEXT := TEMP;
    else
      -- the pointer is pointing the first record in the list.
      COMPLETE_LIST := TEMP;
    end if;

else
  PREV := TEMP;
  SCHEDULE_INPUTS_LIST.NEXT(TEMP);
```

```
                V :~ TEMP;
            end if;
            end loop;
             while SCHEDULE_INPUTS_LIST.NON_EMPTY(TEMP) loop
                if not (OPERATOR_IN_LIST
                             (SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_OPERATOR,
                              AGENDA)) then
                   Exception_Operator := SCHEDULE_INPUTS_LIST.VALUE(TEMP).THE_OPERATOR;
                      raise MISSED_OPERATOR;
                end if;
                   SCHEDULE_INPUTS_LIST.NEXT(TEMP);
            end loop;
            A := AGENDA;
            SCHEDULE_INPUTS_LIST.LIST_REVERSE(A, AGENDA);
       end PROCESS_EDL_AGENDA;
-------------------------------------------------------------------------

   procedure SCHEDULE_WITH_EARLIEST_START
                     (THE_GRAPH : in DIGRAPH.GRAPH;
                      AGENDA : in out SCHEDULE_INPUTS_LIST.LIST;
                      HARMONIC_BLOCK_LENGTH : in INTEGER) is
      -- used to find a feasible schedule for Earliest Start Scheduling Algorithm.
      THE_OP_INFO_LIST : OP_INFO_LIST.LIST;
      MAY_BE_AVAILABLE : SCHEDULE_INPUTS_LIST.LIST;
      H_B_L : INTEGER := HARMONIC_BLOCK_LENGTH;
      L : OP_INFO_LIST.LIST;
      P : OP_INFO;

   begin
      BUILD_OP_INFO_LIST(THE_GRAPH, THE_OP_INFO_LIST);
      L := THE_OP_INFO_LIST;
      -- put all the end nodes, which has no predicessors, into
      -- MAY_BE_AVAILABLE list
      while OP_INFO_LIST.NON_EMPTY(L) loop
        P := OP_INFO_LIST.VALUE(L);
        if NOT(DIGRAPH.V_LISTS.NON_EMPTY(P.PREDICESSORS)) then
           PROCESS_EST_END_NODE(MAY_BE_AVAILABLE, P.NODE);
        end if;
        OP_INFO_LIST.NEXT(L);
      end loop;
      PROCESS_EST_AGENDA(THE_OP_INFO_LIST, MAY_BE_AVAILABLE, AGENDA, H_B_L);
   end SCHEDULE_WITH_EARLIEST_START;
-------------------------------------------------------------------------

   procedure SCHEDULE_WITH_EARLIEST_DEADLINE
                     (THE_GRAPH              : in DIGRAPH.GRAPH;
                      AGENDA                 : in out SCHEDULE_INPUTS_LIST.LIST;
                      HARMONIC_BLOCK_LENGTH : in INTEGER) is

      -- used to find a feasible schedule for Earliest Deadline Scheduling
      -- Algorithm
      THE_OP_INFO_LIST : OP_INFO_LIST.LIST;
      MAY_BE_AVAILABLE : SCHEDULE_INPUTS_LIST.LIST;
```

```
        H_B_L : INTEGER := HARMONIC_BLOCK_LENGTH;
        L : OP_INFO_LIST.LIST;
        P : OP_INFO;

    begin
        BUILD_OP_INFO_LIST(THE_GRAPH, THE_OP_INFO_LIST);
        L := THE_OP_INFO_LIST;
        -- put all the end nodes, which has no predecessors, into
        -- MAY_BE_AVAILABLE list
        while OP_INFO_LIST.NON_EMPTY(L) loop
          P := OP_INFO_LIST.VALUE(L);
          if NOT(DIGRAPH.V_LISTS.NON_EMPTY(P.PREDICESSORS)) then
            PROCESS_EDL_END_NODE(MAY_BE_AVAILABLE, P.NODE);
          end if;
          OP_INFO_LIST.NEXT(L);
        end loop;
        PROCESS_EDL_AGENDA(THE_OP_INFO_LIST, MAY_BE_AVAILABLE, AGENDA, H_B_L);
    end SCHEDULE_WITH_EARLIEST_DEADLINE;
-------------------------------------------------------------------
    procedure CREATE_STATIC_SCHEDULE (THE_GRAPH : in DIGRAPH.GRAPH;
                          THE_SCHEDULE_INPUTS   : in SCHEDULE_INPUTS_LIST.LIST;
                          HARMONIC_BLOCK_LENGTH : in INTEGER) is
        -- creates the static schedule output and prints to "ss.a" file.
        V_LIST : DIGRAPH.V_LISTS.LIST := THE_GRAPH.VERTICES;
        S : SCHEDULE_INPUTS_LIST.LIST := THE_SCHEDULE_INPUTS;
        SCHEDULE   : TEXT_IO.FILE_TYPE;
        OUTPUT : TEXT_IO.FILE_MODE := TEXT_IO.OUT_FILE;
        COUNTER : INTEGER := 1;

        package VALUE_IO is new TEXT_IO.INTEGER_IO(VALUE);
        use VALUE_IO;
        package INTEGERIO is new TEXT_IO.INTEGER_IO(INTEGER);
        use INTEGERIC;

    begin
        TEXT_IO.CREATE(SCHEDULE, OUTPUT, "/n/suns2/work/caps/prototypes/ss.a");
        TEXT_IO.PUT_LINE(SCHEDULE, "with TL; use TL;");
        TEXT_IO.PUT_LINE(SCHEDULE, "with DS_PACKAGE; use DS_PACKAGE;");
        TEXT_IO.PUT(SCHEDULE, "with PRIORITY_DEFINITIONS; ");
        TEXT_IO.PUT_LINE (SCHEDULE, "use PRIORITY_DEFINITIONS;");
        TEXT_IO.PUT_LINE(SCHEDULE, "with CALENDAR; use CALENDAR;");
        TEXT_IO.PUT_LINE(SCHEDULE, "with TEXT_IO; use TEXT_IO;");
        TEXT_IO.PUT_LINE(SCHEDULE, "procedure STATIC_SCHEDULE is");

        while DIGRAPH.V_LISTS.NON_EMPTY(V_LIST) loop
          TEXT_IO.SET_COL(SCHEDULE, 3);
          VARSTRING.PUT(SCHEDULE, DIGRAPH.V_LISTS   S(V_LIST).THE_OPERATOR_ID);
          TEXT_IO.PUT_LINE(SCHEDULE, "_TIMING_ERR   exception;");
          DIGRAPH.V_LISTS.NEXT(V_LIST);
        end loop;
```

```
TEXT_IO.SET_COL(SCHEDULE, 3);
TEXT_IO.PUT_LINE(SCHEDULE, "task SCHEDULE is");
TEXT_IO.SET_COL(SCHEDULE, 5);
TEXT_IO.PUT_LINE(SCHEDULE, "pragma priority (STATIC_SCHEDULE_PRIORITY);");
TEXT_IO.SET_COL(SCHEDULE, 3);
TEXT_IO.PUT_LINE(SCHEDULE, "end SCHEDULE;");
TEXT_IO.NEW_LINE(SCHEDULE);
TEXT_IO.SET_COL(SCHEDULE, 3);
TEXT_IO.PUT_LINE(SCHEDULE, "task body SCHEDULE is");
TEXT_IO.PUT(SCHEDULE, "    PERIOD : constant := ");
INTEGERIO.PUT(SCHEDULE, HARMONIC_BLOCK_LENGTH, 1);
TEXT_IO.PUT_LINE(SCHEDULE, ";");
S := THE_SCHEDULE_INPUTS;
while SCHEDULE_INPUTS_LIST.NON_EMPTY(S) loop
   TEXT_IO.SET_COL(SCHEDULE, 5);
   VARSTRING.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATOR);
   TEXT_IO.PUT(SCHEDULE, "_STOP_TIME");
   INTEGERIO.PUT(SCHEDULE, COUNTER,1);
   TEXT_IO.PUT(SCHEDULE, " : constant := ");
   VALUE_IO.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_STOP, 1);
   TEXT_IO.PUT_LINE(SCHEDULE, ";");
   SCHEDULE_INPUTS_LIST.NEXT(S);
   COUNTER := COUNTER + 1;
end loop;
TEXT_IO.SET_COL(SCHEDULE, 5);
TEXT_IO.PUT_LINE(SCHEDULE, "SLACK_TIME : duration;");
TEXT_IO.SET_COL(SCHEDULE, 5);
TEXT_IO.PUT_LINE(SCHEDULE, "START_OF_PERIOD : time := clock;");
TEXT_IO.PUT_LINE(SCHEDULE, "begin");
TEXT_IO.PUT_LINE(SCHEDULE, "  loop");
TEXT_IO.SET_COL(SCHEDULE, 5);
TEXT_IO.PUT(SCHEDULE, "begin");

S := THE_SCHEDULE_INPUTS;
COUNTER := 1;
while SCHEDULE_INPUTS_LIST.NON_EMPTY(S) loop
   TEXT_IO.SET_COL(SCHEDULE, 7);
   VARSTRING.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATOR);
   TEXT_IO.PUT_LINE(SCHEDULE, ";");
   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT(SCHEDULE, "SLACK_TIME := START_OF_PERIOD + ");
   VARSTRING.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATOR);
   TEXT_IO.PUT(SCHEDULE, "_STOP_TIME");
   INTEGERIO.PUT(SCHEDULE, COUNTER,1);
   TEXT_IO.PUT_LINE(SCHEDULE, " - CLOCK;");
   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "if SLACK_TIME >= 0.0 then");
   TEXT_IO.SET_COL(SCHEDULE, 9);
   TEXT_IO.PUT_LINE(SCHEDULE, "delay (SLACK_TIME);");
   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "else");
```

```
      TEXT_IO.SET_COL(SCHEDULE, 9);
      TEXT_IO.PUT(SCHEDULE, "raise ");
      VARSTRING.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_OPERATOR);
      TEXT_IO.PUT_LINE(SCHEDULE, "_TIMING_ERROR;");
      TEXT_IO.SET_COL(SCHEDULE, 7);
      TEXT_IO.PUT_LINE(SCHEDULE, "end if;");
      SCHEDULE_INPUTS_LIST.NEXT(S);
      if SCHEDULE_INPUTS_LIST.NON_EMPTY(S) then
        -- pointer is pointing to the next record after this.
        TEXT_IO.SET_COL(SCHEDULE, 7);
        TEXT_IO.PUT(SCHEDULE, "delay (START_OF_PERIOD + ");
        VALUE_IO.PUT(SCHEDULE, SCHEDULE_INPUTS_LIST.VALUE(S).THE_START, 1);
        TEXT_IO.PUT_LINE(SCHEDULE, " - CLOCK);");
        TEXT_IO.NEW_LINE(SCHEDULE);
      end if;
      COUNTER := COUNTER + 1;
   end loop;

   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "START_OF_PERIOD := START_OF_PERIOD + PERIOD;");
   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "delay (START_OF_PERIOD - clock);");

   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "exception");
   V_LIST := THE_GRAPH.VERTICES;
   while DIGRAPH.V_LISTS.NON_EMPTY(V_LIST) loop
      TEXT_IO.SET_COL(SCHEDULE, 9);
      TEXT_IO.PUT(SCHEDULE, "when ");
      VARSTRING.PUT(SCHEDULE, DIGRAPH.V_LISTS.VALUE(V_LIST).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, "_TIMING_ERROR =>");
      TEXT_IO.SET_COL(SCHEDULE, 11);
      TEXT_IO.PUT(SCHEDULE, "PUT_LINE(""timing error from operator ");
      VARSTRING.PUT(SCHEDULE, DIGRAPH.V_LISTS.VALUE(V_LIST).THE_OPERATOR_ID);
      TEXT_IO.PUT_LINE(SCHEDULE, """");");
      TEXT_IO.SET_COL(SCHEDULE, 11);
      TEXT_IO.PUT_LINE(SCHEDULE, "START_OF_PERIOD := clock;");
      DIGRAPH.V_LISTS.NEXT(V_LIST);
   end loop;

   TEXT_IO.SET_COL(SCHEDULE, 7);
   TEXT_IO.PUT_LINE(SCHEDULE, "end;");
   TEXT_IO.SET_COL(SCHEDULE, 5);
   TEXT_IO.PUT_LINE(SCHEDULE, "end loop;");
   TEXT_IO.SET_COL(SCHEDULE, 3);
   TEXT_IO.PUT_LINE(SCHEDULE, "end SCHEDULE;");
   TEXT_IO.NEW_LINE(SCHEDULE);
   TEXT_IO.PUT_LINE(SCHEDULE, "begin");
   TEXT_IO.SET_COL(SCHEDULE, 3);
   TEXT_IO.PUT_LINE(SCHEDULE, "null;");
   TEXT_IO.PUT_LINE(SCHEDULE, "end STATIC_SCHEDULE;");
```

334

```
    end CREATE_STATIC_SCHEDULE;

end OPERATOR_SCHEDULER;
```

APPENDIX AB STATIC SCHEDULER LIST STRUCTURE

```
--------------------------------------------------------------------
-- file:      sequence_s.a
-- author:    murat kilic
--            isaac mostov
--            tony davis
-- date:      sep 89
-- modified:  oct 89 by murat kilic
--------------------------------------------------------------------


generic
  type ITEM is private;

  package SEQUENCES is

    type NODE;
    type LIST is access NODE;
    type NODE is
      record
        ELEMENT : ITEM;
        NEXT    : LIST;
      end record;

    BAD_VALUE : exception;

    function EQUAL(L1 : in LIST; L2 : in LIST) return BOOLEAN;

    procedure EMPTY(L : out LIST);

    function NON_EMPTY(L : in LIST) return BOOLEAN;

    function SUBSEQUENCE(L1 : in LIST; L2 : in LIST) return BOOLEAN;

    function MEMBER(X : in ITEM; L : in LIST) return BOOLEAN;

    procedure ADD(X : in ITEM; L : in out LIST);

    procedure REMOVE(X : in ITEM; L : in out LIST);

    procedure LIST_REVERSE(L1 : in LIST; L2 : in out LIST);

    procedure DUPLICATE(L1 : in LIST; L2 : in out LIST);

    function LOOK4(X : in ITEM; L : in LIST) return LIST;
```

```
procedure NEXT(L : in out LIST);

function VALUE(L : in LIST) return ITEM;

end SEQUENCES;
```

```
--------------------------------------------------------------------
-- file:        sequence_b.a
-- author:      murat kilic
--              isaac mostov
--              tony davis
-- date:        sep 89
-- modified:    oct 89 by murat kilic
--------------------------------------------------------------------


with UNCHECKED_DEALLOCATION;

package body SEQUENCES is

  procedure FREE is new UNCHECKED_DEALLOCATION(NODE, LIST);

  function NON_EMPTY(L : in LIST) return BOOLEAN is
  begin
    if L = null then
      return FALSE;
    else
      return TRUE;
    end if;
  end NON_EMPTY;

  procedure NEXT(L : in out LIST) is
  begin
    if L /= null then
      L := L.NEXT;
    end if;
  end NEXT;

  function LOOK4(X : in ITEM; L : in LIST) return LIST is
    L1 : LIST := L;
  begin
    while NON_EMPTY(L1) loop
      if L1.ELEMENT = X then
        return L1;
      end if;
      NEXT(L1);
    end loop;
    return null;
  end LOOK4;
```

```
procedure ADD(X : in ITEM; L : in out LIST) is
-- ITEM IS ADDED TO THE HEAD OF THE LIST
  T : LIST := new NODE;
begin
  T.ELEMENT := X;
  T.NEXT := L;
  L := T;
end ADD;

function SUBSEQUENCE(L1 : in LIST; L2 : in LIST) return BOOLEAN is
  L : LIST := L1;
begin
  while NON_EMPTY(L) loop
    if not MEMBER(VALUE(L), L2) then
      return FALSE;
    end if;
    NEXT(L);
  end loop;
  return TRUE;
end SUBSEQUENCE;

function EQUAL(L1 : in LIST; L2 : in LIST) return BOOLEAN is
begin
  return (SUBSEQUENCE(L1, L2) and SUBSEQUENCE(L2, L1));
end EQUAL;

procedure EMPTY(L : out LIST) is
begin
  L := null;
end EMPTY;

function MEMBER(X : in ITEM; L : in LIST) return BOOLEAN is
begin
  if LOOK4(X, L) /= null then
    return TRUE;
  else
    return FALSE;
  end if;
end MEMBER;

procedure REMOVE(X : in ITEM; L : in out LIST) is
  CURR : LIST := L;
  PREV : LIST := null;
  TEMP : LIST := null;
begin
  while NON_EMPTY(CURR) loop
    if VALUE(CURR) = X then
      TEMP := CURR;
      NEXT(CURR);
      FREE(TEMP);
      if PREV /= null then
```

339

```
          PREV.NEXT := CURR;
        else
          L := CURR;
        end if;
      else
        PREV := CURR;
        NEXT(CURR);
      end if;
    end loop;
  end REMOVE;

  procedure LIST_REVERSE(L1 : in LIST; L2 : in out LIST) is
    L : LIST := L1;
  begin
    EMPTY(L2);
    while NON_EMPTY(L) loop
      ADD(VALUE(L), L2);
      NEXT(L);
    end loop;
  end LIST_REVERSE;

  procedure DUPLICATE(L1 : in LIST; L2 : in out LIST) is
    TEMP : LIST;
    L    : LIST := L1;
  begin
    EMPTY(L2);
    while NON_EMPTY(L) loop
      ADD(VALUE(L), TEMP);
      NEXT(L);
    end loop;
    LIST_REVERSE(TEMP, L2);
  end DUPLICATE;

  function VALUE(L : in LIST) return ITEM is
  begin
    if NON_EMPTY(L) then
      return L.ELEMENT;
    else
      raise BAD_VALUE;
    end if;
  end VALUE;

end SEQUENCES;
```

340

APPENDIX AD STATIC SCHEDULER TOPOLOGICAL SORTER

```
-------------------------------------------------------------------
-- file:      t_sort_s.a
-- author:    murat_kilic
-- date:      oct 89
-- modified:  dec 89 by murat kilic
-------------------------------------------------------------------


with FILES;use FILES;
package TOPOLOGICAL_SORTER is

  procedure TOPOLOGICAL_SORT
                (G                 : in DIGRAPH.GRAPH;
                 PRECEDENCE_LIST : in out DIGRAPH.V_LISTS.LIST);

end TOPOLOGICAL_SORTER;
```

APPENDIX AE STATIC SCHEDULER TOPOLOGICAL SORTER

```
-----------------------------------------------------------------
-- file:      t_sort_b.a
-- author:    murat kilic
-- date:      oct 89
-- modified:  nov 89 by murat kilic
-----------------------------------------------------------------


with TEXT_IO;
with FILES; use FILES;

package body TOPOLOGICAL_SORTER is

  -- This package determines the precedence order in which operators must
  -- execute in the final schedule.  This information is determined
  -- from the graph.

  procedure TOPOLOGICAL_SORT (G: in DIGRAPH.GRAPH;
                  PRECEDENCE_LIST: in out DIGRAPH.V_LISTS.LIST) is

    -- This procedure determines which operators in the graph must
    -- be executed before another.

  Q : DIGRAPH.V_LISTS.LIST;

  begin
    DIGRAPH.T_SORT(G,PRECEDENCE_LIST);
    Q := PRECEDENCE_LIST;
  end TOPOLOGICAL_SORT;

end TOPOLOGICAL_SORTER;
```

APPENDIX AF DYNAMIC SCHEDULER

```
---------------------------------------------------------------------
-- file:     dynamic_scheduler.a
-- author:   frank palazzo
-- date:     dec 89
-- modified: dec 89 by laura j. white
---------------------------------------------------------------------


with TEXT_IO; use TEXT_IO;
procedure DYNAMIC_SCHEDULER is
  NON_CRITS   :  FILE_TYPE;
  DSV3        :  FILE_TYPE;
  IN_STRING   :  STRING(1..72);
  LAST        :  NATURAL;
begin
  OPEN(NON_CRITS, IN_FILE, "/n/suns2/work/caps/prototypes/non_crits");
  CREATE(DSV3, OUT_FILE, "/n/suns2/work/caps/prototypes/ds.a");
  PUT_LINE(DSV3, "with TL; use TL;");
  PUT_LINE(DSV3, "package DS_PACKAGE is");
  PUT_LINE(DSV3, "  task DYNAMIC_SCHEDULE is");
  -- system defined priority for dynamic schedule
  PUT_LINE(DSV3, "    pragma priority (1);");
  PUT_LINE(DSV3, "  end DYNAMIC_SCHEDULE;");
  PUT_LINE(DSV3, "end DS_PACKAGE;");
  NEW_LINE(DSV3);
  PUT_LINE(DSV3, "package body DS_PACKAGE is");
  PUT_LINE(DSV3, "  task body DYNAMIC_SCHEDULE is");
  PUT_LINE(DSV3, "  begin");
  PUT_LINE(DSV3, "    delay (1.0);");
  while not END_OF_FILE(NON_CRITS) loop
    begin
      GET_LINE(NON_CRITS, IN_STRING, LAST);
      PUT(DSV3, "      ");
      for INDEX in 1..LAST loop
        PUT(DSV3, IN_STRING(INDEX));
      end loop;
      PUT_LINE(DSV3, ";");
    end;
  end loop;
  PUT_LINE(DSV3, "  end DYNAMIC_SCHEDULE;");
  PUT_LINE(DSV3, "end DS_PACKAGE;");
end DYNAMIC_SCHEDULER;
```

LIST OF REFERENCES

1. Booch, G., *Software Engineering with Ada*, 2d ed., Benjamin/Cummings, 1987.

2. Schach, S. R., *Software Engineering*, Aksen Associates, 1990.

3. Lamb, D. A., *Software Engineering Planning for Change*, Prentice Hall, 1988.

4. Boehm, B. W., "A Spiral Model of Software Development and Enhancement", *ACM SIGSOFT Software Engineering Notes*, v. 11, no. 4, pp. 14-26, August 1986.

5. Luqi, "Software Evolution Through Rapid Prototyping", *Computer*, v. 22, no. 5, pp. 13-25, May 1989.

6. Boehm, B., "Verifying and Validating Software Requirements and Design Specifications", *IEEE Software*, v. 1 no. 1, January 1984.

7. Luqi, "Handling Timing Constraints in Rapid Prototyping", *Proceedings of the 22nd Annual Hawaii International Conference on System Sciences*, IEEE Computer Society, pp. 4-17-424, January 1989.

8. Tanik, M. M. and Yeh, R. T., "Rapid Prototyping in Software Development", *Computer*, v. 22, n. 5, pp. 9-10, May 1989.

9. Luqi, *Rapid Prototyping for Large Software System Design*, Ph.D. Dissertation, University of Minnesota, Minneapolis, Minnesota, May 1986.

10. Thorstenson, R. K., *A Graphical Editor for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

11. Porter, S. W., *Design of a Syntax Directed Editor for PSDL*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

12. Douglas, B. S., *A Conceptual Level Design of a Design Database for the Computer-Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

13. Galik, D., *A Conceptual Design of a Software Base Management System For the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

14. Altizer, C., *Implementation of a Language Translator for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

15. Marlowe, L., *A Scheduler for Critical Time Constraints*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

16. Kilic, M., *Static Schedulers for Embedded Real-Time Systems*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1989.

17. Wood, M. B., *Run-Time Support for Rapid Prototyping*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

18. Ambler, A. L. and Burnett, M. M., "Influence of Visual Technology on the Evolution of Language Environments", *Computer*, v. 22, n.10, pp. 9-22, October 1989.

19. MacLennan, B. J., *Principles of Programming Languages Design, Evaluation, and Implementation*, 2d ed., Holt, Rinehart and Winston, 1987.

20. Naval Postgraduate School NPS52-89-026, *Issues in Language Support for Rapid Prototyping*, by Luqi and Berzins, March 1989.

21. Rochkind, M. J., *Advanced UNIX Programming*, Prentice-Hall, 1985.

22. Kaplan, S. M. and others, "An architecture for Tool Integration", pp. 112-125, in *Advanced Programming Environments*, Springer-Verlag, 1986.

23. Raum, H. G., *Design and Implementation of an Expert User Interface for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

24. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, 1987.

25. Darnell, P. A. and Margolis, P. E., *Software Engineering in C*, Springer-Verlag, 1988.

26. Kernighan, B. W. and Ritchie, D. M., *The C Programming Language*, 2d ed., Prentice-Hall, 1988.

27. Naval Postgraduate School NPS52-89-028, *Graphical Support for Reducing Information Overload in Rapid Prototyping*, by Luqi and Barnes P. D, March 1989.

28. Reps, T. W. and Teitelbaum, T., *The Synthesizer Generator: A System for Constructing Language-Based Editors*, Springer-Verlag, 1989.

29. Reps, T. W. and Teitelbaum, T., *The Synthesizer Generator Reference Manual*, 3d ed., Springer-Verlag, 1989.

30. Parnas, D. L., "Enhancing Reusability with Information Hiding", *ITT Proceedings of the Workshop on Reusability in Programming*, pp. 240-247, 1983.

31. Conn, R., *The Ada Software Repository and the Defense Data Network*, Zoetrope Publishing Co., Inc., New York, NY, 1987.

32. Johnson, R. E. and Foote, B., "Designing Reusable Classes", *Journal of Object-Oriented Programming*, June/July 1988.

33. Matsumoto, Y., "A Software Factory: An Overall Approach to Software Production", *Tutorial:Software Reusability*, Computer Society Press of the IEEE, 1987.

34. Burton, B. A., "The Reusable Software Library,", *IEEE Software*, pp. 25-32, July 1987.

35. Prieto-Diaz, R. and Freeman, P., "Classifying Software for Reusability", *IEEE Software*, pp.6-16, January 1987.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Dudley Knox Library   2
   Code 0142
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Director of Research Administration   1
   Attn: Prof. Howard
   Code 012
   Naval Postgraduate School
   Monterey, California 93943-5100

4. Chairman   1
   Code 52
   Naval Postgraduate School
   Monterey, California 93943-5100

5. Chief of Naval Research   1
   800 N. Quincy Street
   Arlington, Virginia 22217-5000

6. Center for Naval Analysis   1
   4401 Ford Avenue
   Alexandria, Virginia 22302-0268

7. National Science Foundation   1
   Division of Computer and Computation Research
   Attn: Tom Keenan
   Washington, D.C. 20550

8. Ada Joint Program Office   1
   OUSDRE(R&AT)
   Pentagon
   Washington, D.C. 20301

9. Naval Sea Systems Command   1
   Attn. CAPT Joel Crandall
   National Center #2, Suite 7N06
   Washington, D. C. 22202

10. Naval Sea Systems Command   1
    Attn. CAPT A. Thompson
    National Center #2, Suite 7N06
    Washington, D. C. 22202

11. Commanding Officer   1
    Naval Research Laboratory
    Code 5150
    Attn. Dr. Elizabeth Wald
    Washington, D.C. 20375-5000

12. Navy Ocean System Center   1
    Attn. Linwood Sutton, Code 423
    San Diego, California 92152-5000

13. Navy Ocean System Center   1
    Attn. Les Anderson, Code 413
    San Diego, California 92152-5000

14. Office of Naval Research   1
    Computer Science Division, Code 1133
    Attn. Dr. Van Tilborg
    800 N. Quincy Street
    Arlington, Virginia 22217-5000

15. Office of Naval Research   1
    Computer Science Division, Code 1133
    Attn. Dr. R. Wachter
    800 N. Quincy Street
    Arlington, Virginia 22217-5000

16. Office of Naval Research   1
    Applied Mathematics and Computer Science, Code 1211
    Attn. Mr. J. Smith
    800 N. Quincy Street
    Arlington, Virginia 22217-5000

17. Defense Advanced Research Projects Agency (DARPA)   1
    Integrated Strategic Technology Office (ISTO)
    Attn. Dr. B. Boehm
    1400 Wilson Boulevard
    Arlington, Virginia 22209-2308

18. Defense Advanced Research Projects Agency (DARPA)   1
    Director, Naval Technology Office
    1400 Wilson Boulevard
    Arlington, Virginia 22209-2308

19. Defense Advanced Research Projects Agency (DARPA)   1
    Director, Prototype Projects Office
    1400 Wilson Boulevard
    Arlington, Virginia 22209-2308

20. Defense Advanced Research Projects Agency (DARPA)   1
    Director, Tactical Technology Office
    1400 Wilson Boulevard
    Arlington, Virginia 22209-2308

21. Chief of Naval Operations   1
    Attn: Dr. R. M. Carroll (OP-01B2)
    Washington, D.C. 20350

22. Chief of Naval Operations   1
    Attn: Dr. Earl Chavis (OP-162)
    Washington, D.C. 20350

23. Naval Surface Warfare Center   1
    Code K54
    Attn: Dr. William McCoy
    Dahlgren, Virginia 22448

24. Naval Surface Warfare Center   1
    Code U33
    Attn: Phil Hwang
    Silver Spring, Maryland 20903-5000

25. Professor Luqi   1
    Code 52Lq
    Naval Postgraduate School
    Computer Science Department
    Monterey, California 93943-5100